MODELADO Y SOPORTE DE LAS TECNOLOGÍAS DE LA WEB 2.0 A PATRONES Y PRINCIPIOS DE RIA

JAVID JALALI MUÑIZ

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Sistemas Inteligentes

Curso 2011 - 2012
Convocatoria de junio
Calificación: Sobresaliente - 9

Director:
Antonio Navarro Martín

Autorización de Difusión

JAVID JALALI MUÑIZ

19 de junio de 2012

El/la abajo firmante, matriculado/a en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: “Modelado y soporte de las tecnologías de la web 2.0 a patrones y principios de RIA”, realizado durante el curso académico 2011-2012 bajo la dirección de D. Antonio Navarro Martín en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

La expansión de la web afecta cada vez a más aspectos de las vidas de las personas y en mayor medida, lo cual aumenta proporcionalmente la importancia de la experiencia que tengan las personas en ella, convirtiéndose en muchos casos en un factor determinante para el éxito de un sitio o aplicación web. Así nacen las *Rich Internet Applications* (RIA), aplicaciones web interactivas y muy dinámicas nacidas en la web 2.0 cuya principal característica es la semejanza de sus mecanismos de interfaz gráfica con los de las aplicaciones tradicionales de escritorio. En este trabajo se estudian los aspectos que diferencian a las RIA con aplicaciones web menos innovadoras y se presentan las principales tecnologías que soportan el desarrollo de este tipo de sistemas. Posteriormente se explica el estado del arte de las notaciones de modelado y los patrones de desarrollo que existen actualmente para facilitar el modelado de RIA y se propone una extensión a UML WAE para modelar las tecnologías objeto de estudio. Finalmente se realiza un estudio detallado del soporte que dan estas tecnologías a uno de los principales catálogos de patrones, el propuesto por Scott y Neil, y se presentan las conclusiones obtenidas.

Palabras clave

RIA, Patrones, Diseño, Modelado, Web 2.0, AJAX, Flex, JSF, Javascript, ExtJS, GWT, HTML5, UML

Resumen en inglés

The expansion of the World Wide Web affects people's lifes in many aspects. This quality makes the user experience of the web proportionally important and determining factor for the success of a site or web application. That's the reason for the creation of Rich Internet Applications (RIA). These are interactive web applications that are very dynamic and are created from the Web 2.0.

Their main feature is the similarity of their graphic interface mechanisms with traditional web applications. This project studies the aspects that make the RIA different from other less innovative web applications and expose the main technologies that are developed in these kind of systems. We will explain the state of the art model annotations and design patterns that are used to make the RIA modeling easier and propose a UML WAE extension to model the technologies that are the object of study.

Finally there is a detailed study of the support these technologies give to the Scott and Neil patterns and a presentation of the final conclusions.

Keywords

RIA, Patterns, Design, Modelling, Web 2.0 AJAX, Flex, JSF, Javascript, ExtJS, GWT, HTML5, UML

Índice de contenidos

Agradecimientos

Quisiera ser agradecido con todas las personas que, de una u otra forma, me han ayudado a completar este ciclo tan importante en mi vida que, en realidad, no es más que el comienzo de uno nuevo en el que espero seguir contando con el apoyo, sinceridad y comprensión que tan importante ha sido para mí durante este tiempo.

Iris, a la que he tenido en los mejores y en los peores momentos y ha sabido aconsejarme, ayudarme y proporcionarme la claridad y la motivación que he necesitado. Ha sido el público sonriente de mis primeros ensayos de presentaciones, pese a lo tedioso que fuera el tema y me ha acompañado en cada uno de los días que me han llevado a escribir estas líneas. Gracias, Iris.

Antonio, a quien debo agradecer su paciencia, comprensión, implicación, sabiduría y afán por hacer las cosas bien. Gracias Antonio por cada reunión, cada email y cada conversación desde hace tantos años ya.

Debo agradecer también a todas las personas que forman y han formado parte de la institución que me ha convertido en el profesional que soy hoy, ya que la universidad es un organismo vivo gracias a su colaboración. Gracias.

Finalmente agradezco a mi familia, compañeros de trabajo y amigos, especialmente a mi madre, a Juanjo y a Álex respectivamente ya que me han apoyado, aconsejado y alentado a dar siempre lo mejor de mí.

Capítulo 1 Introducción / Motivación

En los últimos años, la web ha tenido una creciente expansión de forma horizontal y vertical. La expansión horizontal simboliza el crecimiento multidisciplinar: son cada vez más las nuevas áreas, servicios y campos los que se ven afectados y, además, ha propiciado la creación de nuevos tipos de negocio, de relaciones y de comunicación. La expansión vertical de la web hace referencia a la magnitud de la influencia que tiene en cada uno de los aspectos a los que afecta, llegando al punto de reinventar modelos de negocio como el discográfico. Esta influencia afecta notoriamente a la informática gracias al carácter global de la Web y al esfuerzo de la comunidad científica.

La web es un instrumento importante que crea nuevas vías de comunicación, haciendo más sencillas las relaciones entre personas, empresas y otros organismos, como por ejemplo, la administración pública.

El éxito de las redes sociales es un ejemplo de la influencia en la comunicación interpersonal, pero los modelos de relación *business-to-business* (B2B) a través de internet también aportan grandes ventajas para las empresas implicadas en ellos.

Aunque las relaciones entre las administraciones públicas a través de la web aún no son tan maduras como las relaciones B2B o interpersonales, se están impulsando muchas medidas que cambien esta situación, como la modernización de la justicia o la digitalización de los registros civiles. En cambio, las relaciones entre las administraciones públicas con empresas y personas están más avanzadas gracias a avances como el certificado digital que permite identificar a una persona o empresa en una aplicación Web de manera inequívoca.

Finalmente, la relación que más impulsa el avance de las tecnologías web es la comercial, aquella que se da entre negocios y personas: el comercio online.

Como fuente de ingresos que son, las tiendas virtuales han sido objetivo de inversión para la mejora a nivel de diseño, marketing, usabilidad y accesibilidad para conseguir que la experiencia del usuario en la web sea óptima, lo cual es fundamental para el éxito: la venta.

Los mismos cánones que afectan a la impresión que producen en las personas factores tales como la apariencia de una tienda física también aplican a las versiones virtuales de éstas: la experiencia que tenga el usuario, por ejemplo, en una tienda online afecta en gran medida al

momento en el que tenga que introducir su número de tarjeta de crédito o su contraseña de PayPal para realizar un pago.

Cuando se visitan sitios en los que aparecen errores en el proceso de compra, su apariencia es arcaica, la búsqueda es inexacta o es complicado acceder a cierto contenido de interés, se hace más complicado que el usuario confíe en que su dinero se va a poner en buenas manos, que su información personal va a ser tratada con rigor, o que su consulta va a ser respondida con celeridad y profesionalidad.

Esta tendencia se propaga de las tiendas virtuales a todo tipo de páginas y aplicaciones web. El aumento del grado de implicación de la web en la sociedad tanto a nivel administrativo, económico, cultural y comunicativo aumenta la competitividad de la imagen y los servicios que se ofrecen en ésta y, dada la interacción bidireccional existente entre ella y los usuarios, cuanto más intuitiva, sencilla y agradable sea su experiencia más posibilidades existen de lograr el éxito sea cual sea éste: vender, comunicar, enseñar, obtener información...

Aunque la web es relativamente joven, sus usuarios suelen estar acostumbrados a trabajar en entornos en los que existe software, como aplicaciones de escritorio, cuyo comportamiento es familiar, predecible y no siempre está basado en el flujo entre pantallas, sino que muchas veces los cambios realizados en una misma pantalla no conllevan la navegación a otra. En cambio, en la web tradicional esto no es así ya que el flujo entre páginas es, en general, indispensable para realizar procesos.

Este es el primer y más importante objetivo de la mejora de la experiencia del usuario en la web: imitar el comportamiento que tienen las aplicaciones de escritorio en las aplicaciones online, así como aumentar la velocidad con la que se obtiene *feedback* de una página para aumentar el rendimiento y evitar que los usuarios queden con frecuencia en situaciones de incertidumbre en las que duden si una página está funcionando correctamente o no.

Las aplicaciones web que buscan la mejora de la experiencia del usuario tal y como se ha descrito anteriormente se denominan *Rich Internet Applications (RIA)*.

En el desarrollo de RIA existen una serie de situaciones que aparecen reiteradamente y que ya han sido objeto de estudio por parte de distintos autores y se han presentado las distintas soluciones tipo para manejarlas adecuadamente: tal y como sucede en el desarrollo de software tradicional, para el desarrollo de RIA también existen patrones de diseño [1, 2, 3] que se explicarán más adelante a lo largo de este trabajo.

Los patrones de diseño se han convertido en catálogos de buenas prácticas en el desarrollo de software. Sin embargo, debido a la reciente aparición de las tecnologías RIA cabe preguntarse hasta qué punto dichas tecnologías son capaces de soportar a los patrones de diseño RIA. Esta característica puede ser fundamental a la hora de elegir tecnologías RIAS específicas para diseños concretos.

Además, a la hora de diseñar aplicaciones RIA surge el problema de las limitaciones existentes en las notaciones de diseño web para soportar las construcciones RIA. Algunas notaciones que se basan en modelar la aplicación final, en vez del código [4, 5, 6] han hecho extensiones, pero en la práctica, dichas notaciones no son utilizadas en la industria. Por otro lado, UML, centrado en modelar el código, no soporta de manera estándar las construcciones RIA. En la actualidad hay extensiones para aplicaciones web que conciben un uso de UML basado en el proceso unificado de desarrollo, como por ejemplo UML WAE [7], pero dichas extensiones no proporcionan soporte a las construcciones RIA.

Esta situación de cierta incertidumbre es complicada para las personas responsables de elegir una tecnología de implementación de RIA y una herramienta o notación de modelado para este tipo de aplicaciones, ya que no hay trabajos que relacionen las tecnologías RIA con los patrones ni extensiones para el estándar de modelado de aplicaciones empresariales. De estos problemas surge la motivación y elaboración de este trabajo, que es de un gran interés debido a la repercusión que tienen este tipo de aplicaciones en la actualidad. Con él, se pretende dar respuesta a los interrogantes anteriormente mencionados.

En el próximo capítulo se estudian las diferencias que existen entre las aplicaciones web tradicionales y las RIA, para después presentar cinco de las principales tecnologías actuales susceptibles de ser utilizadas para el desarrollo de este tipo de aplicaciones: *Javascript + HTML* (representado por ExtJS), *Adobe Flex*, *Google Web Toolkit*, *HTML5*, y *Java Server Faces*. Para cada una de ellas hay referencias a ejemplos reales y muestras de cómo son los lenguajes de programación que las implementan.

En el capítulo 3 se expone el estado del arte de las notaciones y de los patrones de diseño para RIA. Posteriormente en el capítulo 4 se define una extensión para WAE basada en el estudio de las interacciones y peculiaridades de las tecnologías presentadas en el capítulo 2.

El capítulo 5 analiza el soporte de estas tecnologías para uno de los principales catálogos de patrones de RIA [3] y finalmente, en el capítulo 6, se presentan las conclusiones y trabajo futuro.

Cabe destacar que las investigaciones llevadas a cabo en este trabajo son del interés del proyecto *Arquitecturas Avanzadas en Campus Virtuales*, AACV, TIN TIN2009-14317-C03-01. Este proyecto tiene por objetivo proporcionar una arquitectura de campus virtuales de nueva generación capaces de responder a las necesidades de sus usuarios. En este contexto, el uso de patrones y tecnologías RIA puede ser de suma utilidad.

Capítulo 2 Tecnologías de la Web 2.0

En este capítulo se dará una visión detallada del estado del arte de las tecnologías seleccionadas para el estudio en el presente trabajo. Primero veremos las principales tecnologías RIA para dar paso a la presentación detallada de cada una de ellas.

2.1 Filosofía de interacción “Request / Response” vs AJAX

El método más antiguo y básico para la visualización de contenido web es el protocolo de petición y respuesta, *request and response*[8]. Esta forma de comunicación se da entre pares de computadores y funciona de la siguiente manera:

1. El computador A *realiza una petición* de datos a un computador B.
2. El computador B *recibe la petición y responde* (si procede) al computador A.

Como se puede ver, se trata de una comunicación *síncrona*.

En internet, estos dos computadores son el cliente y el servidor y será la forma en la que sean denominados de aquí en adelante. Llamaremos *cliente* a aquel computador que, en circunstancias normales, será utilizado por una persona y realizará peticiones a través de internet a *servidores*, que son computadores que sólo son operados por personas para labores de mantenimiento, control, etc. El cliente realiza peticiones de datos al servidor que serán contestadas y mostradas al usuario mediante un *navegador web*. Un ejemplo de esta interacción se podría entender fácilmente con el siguiente ejemplo:

1. Un usuario escribe en la barra de direcciones de su navegador la dirección http://www.example.com y presiona el botón requerido por dicho navegador para ir a esa dirección. Al pulsarlo, el cliente interpreta el deseo del usuario de visualizar esa web y lo convierte a una petición http que envía, simplificando el proceso, al servidor que aloja a dicha web.
2. El servidor donde se aloja el sitio web procesa esa petición y envía a la dirección que envío originalmente la petición el código HTML correspondiente a la página de inicio.

3. El cliente recibe ese código que asciende por las capas de transmisión de datos hasta el navegador, que interpreta el código HTML y lo convierte en información entendible para el usuario.

Durante todo el tiempo transcurrido entre el paso 1 y el 3 del ejemplo anterior, el contenido que se estuviera mostrando en ese momento en el navegador (en la pestaña o instancia en cada caso) ha quedado bloqueado a la espera de recibir la respuesta del servidor debido al sincronismo de este método. El sincronismo es fundamental en el desarrollo de aplicaciones web ya que en muchas ocasiones los usuarios deben ser notificados de manera activa cuando la acción que han tomado ya ha concluido. Por ejemplo, cuando un usuario envía un formulario de contacto, espera recibir un mensaje que le indique que el mensaje se ha enviado correctamente.

No obstante, es inmediato pensar que, en este mismo ejemplo, sería igualmente útil recargar tan sólo una parte de la web para mostrar un simple mensaje de estado. Al recargar sólo una parte de la página se evitaría bloquear el uso de la página: en este caso podríamos pensar que el bloqueo sería muy corto debido a la sencillez del proceso, pero imaginemos que en lugar de un formulario de contacto se ha realizado una petición que devuelva un listado con el stock de un almacén: ya no sería igual de sencillo y rápido. Pero no sólo se daría libertad para realizar más acciones en la misma página desde la cual se ha realizado la petición: otra ventaja sería evitar el envío de la misma información una y otra vez: pensemos en cabeceras, menús, pies de página… Esto haría la comunicación mucho más ligera y, por lo tanto más fluida entre el cliente y el servidor.

Los métodos que hacen posible la carga asíncrona de una página web, es decir, de la carga síncrona de tan sólo un área de una página web mientras el resto permanece inalterado, se suelen denominar de forma incorrecta AJAX. Estas siglas en inglés, *Asynchronous Javascript and XML* [9] no encajan con el concepto de conjunto de técnicas con el que erróneamente se asocia ya que, en realidad, se trata de una sola de éstas técnicas como veremos a continuación.

El primer método que permitió este tipo de comportamiento en un navegador web fue el uso de *Applets de Java* [10]. Los Applets de java son aplicaciones que se compilan en el cliente y que, entre otras posibilidades, permite comunicarse con un servidor sin interferir con la visualización del resto de la página. Esto es posible gracias a que los Applets se descargan, compilan y ejecutan gracias a *plugins* o complementos software que se instalan para extender la

funcionalidad de los navegadores. Los applets encajan en la definición de cliente pesado ya que la gran parte del cómputo se realiza en el cliente.

El uso de esta tecnología también tiene ciertos inconvenientes:

1. Representa un riesgo para la seguridad del computador cliente ya que puede permitirse la ejecución de código malicioso o bien puede tener la funcionalidad limitada en caso de no poseer un certificado.
2. Requiere el plugin anteriormente citado. Esto es, que no viene instalado por defecto en el navegador y puede darse el caso de que usuarios sin privilegios en el computador no puedan instalarlo.
3. La primera vez que se ejecuta una aplicación es más lenta. Conlleva cierta latencia.

Otro método para la carga asíncrona y el que mejor puede ejemplificar esta alternativa a la carga síncrona es el uso de los elementos *HTML frame y frameset* [11]. Los frames permiten a los autores de páginas web subdividir la ventana activa en dos o más ventanas independientes. Cada una de estas ventanas puede mostrar un documento HTML diferente y están sujetas al uso de sus propios scrolls.

La implementación de los mecanismos que permiten esta técnica hacen muy simple dividir una página en, por ejemplo, tres áreas independientes: la cabecera con el título del sitio, el menú de navegación y el contenido en sí. Utilizando código Javascript las ventanas pueden enviarse información entre sí y pueden capturarse eventos del usuario, lo cual hace posible que un clic en un ítem de menú contenido en un frame dispare la carga de cierto documento HTML en otro.

Si bien esta técnica es sencilla y antigua, está ampliamente desaconsejada debido a:

1. Dificultan la accesibilidad.
2. El historial del navegador puede no funcionar correctamente
3. El código Javascript que se utiliza para comunicar a los frames entre sí puede complicarse mucho.
4. No están soportados por todos los navegadores.
5. Normalmente sólo puede añadirse como favorita la página principal (la que contiene la definición de los frames) ya que, en otro caso, puede que se añada sólo el contenido de uno de los frames en lugar del conjunto.

6. Por el mismo motivo sólo pueden promocionarse en la web enlaces a la página principal.
7. Los enlaces a páginas fuera del sitio se abren dentro de uno de los frames.
8. El uso de scrolls disminuye el área útil de la web y la hace más difícil de visualizar en dispositivos móviles.

Similar a la aproximación de Applets es la aproximación de *Adobe Flash* [12]. Flash es una tecnología que es ampliamente conocida por su prácticamente total presencia en anuncios, animaciones y reproductores de video en internet. Pero es mucho más que eso: Flash permite, además de la generación dinámica de animaciones audiovisuales, el desarrollo de aplicaciones completas que funcionan tanto incrustadas en una página web como ejecutadas directamente en un computador. Como los Applets de Java, necesita un plugin para poder reproducirse pero está mucho más extendido que éstos.

El lenguaje de programación de Flash, ActionScript permite, entre otras funcionalidades, la conexión en segundo plano con un servidor, lo cual hace de esta tecnología otra alternativa para la implementación de la carga asíncrona. La versión más moderna y enfocada a RIA de Flash se denomina *Flex* [13].

Después de enumerar las distintas alternativas anteriores que ofrecen posibilidades diversas de implementar la carga asíncrona dependiendo de los requisitos de nuestra aplicación, pasaremos a la que da nombre a esa sección: AJAX.

AJAX es un conjunto de tecnologías que se utilizan en conjunto para elaborar sitios web que funcionen mediante la carga asíncrona de contenido [8]. Por si fuera insuficiente el malentendido producido por el uso erróneo de la denominación de este conjunto de tecnologías para englobar el proceso de carga asíncrona de contenido web, existe también un malentendido a con sus siglas también cuando se refieren a la propia tecnología. El nombre AJAX hace referencia al uso de XML como formato de intercambio de datos pero en el desarrollo web, cada vez es más frecuente el uso de otros lenguajes como *JSON* [14, 15]. Las tecnologías que permiten el intercambio de información en AJAX permiten que ésta sea a través de HTML, XML, JSON, texto plano e incluso cualquier lenguaje dependiente del contexto. Además, la interfaz `XMLHttpRequest` también *permite la comunicación síncrona*.

La forma en la que se consigue la carga asíncrona con AJAX es mediante el uso de la interfaz `XMLHttpRequest`, un objeto que implementan casi todos los navegadores actuales y al

que se accede a través de código Javascript. Entonces, podemos acceder a este objeto después de prácticamente cualquier que interacción que tenga el usuario con la web, ya que este lenguaje permite capturar dichos eventos. Este objeto permite el envío de datos a una URL determinada y asignar a una función Javascript como *callback* de esta llamada. Esto es, cuando el servidor responde a esta llamada se ejecuta el *callback* que siempre debe tener un parámetro que recibe la respuesta en formato texto del servidor. La ejecución de ese callback en código Javascript se realiza en cualquier momento *y de manera independiente a lo que esté haciendo el usuario en ese momento.* Además, el código Javascript permite manipular el *DOM* [16] de un documento HTML por lo que dentro de un callback es posible acceder a cualquier elemento que se esté visualizando y modificarlo. Esto representa una gran diferencia a las tres aproximaciones anteriores que mantenían un área de la página como dominio. En este caso el dominio es la página entera.

Un ejemplo del uso de AJAX. Imaginemos una página con dos menús desplegables o combo boxes donde el contenido del segundo depende de la selección que se ha hecho sobre el primero:

1. Un usuario hace clic sobre un ítem del primer combo box.
2. El capturador de ese evento realiza una llamada asíncrona mediante `XMLHttpRequest` al servidor, pasándole como argumentos el identificador del ítem seleccionado por el usuario y el nombre de la función *callback* que va a atender la respuesta.
3. El *callback* anterior se ejecuta al recibir respuesta y manipula el código HTML para añadir los ítems recibidos.
4. El usuario hace clic sobre el segundo combo box y encuentra las opciones adecuadas.

Si el usuario fuera muy rápido al hacer clic y/o la llamada al servidor fuera muy lenta, habría un momento en el que el combo box aparecería vacío o relleno con sólo algunos de los datos recibidos.

2.2 Tecnologías

En el apartado anterior se han podido ver las diferencias entre las dos aproximaciones de comunicación entre cliente y servidor (síncrona y asíncrona). También se han visto distintas implementaciones que existen para poder llevar a cabo el método asíncrono. En este apartado

veremos las distintas tecnologías que se estudian en este trabajo y que son las más relevantes a día de hoy para lograr el desarrollo de aplicaciones web asíncronas.

2.2.1 Javascript y HTML

Como ya se ha explicado, los navegadores de internet pueden utilizar la interfaz `XMLHttpRequest` a través de código Javascript para realizar llamadas asíncronas. Según esta premisa cualquier desarrollador puede escribir su propio código y, estudiando el API proporcionada, realizar llamadas asíncronas al servidor.

Para ello se deben capturar los eventos deseados que son producidos por el usuario en el navegador y, a consecuencia de algunos de estos, se produciría la llamada asíncrona. Mientras el usuario continuara realizando operaciones en la página se desencadenarían una serie de acciones en paralelo que incluirían la recepción del mensaje por parte del servidor, su procesamiento en la capa de negocio y/o integración y la transmisión de la respuesta.

En términos generales, esta respuesta sería recibida por el navegador cliente y seguiría siendo procesada en paralelo al usuario. En un momento dado, el procesamiento de la respuesta podría conllevar ciertos cambios en la interfaz que se realizarían siguiendo el orden de ejecución descrito. De esta manera el usuario vería un cambio en la página que está visitando más parecido a una transformación que a una recarga de la página.

Todo este proceso es largo y tedioso debido a que Javascript es un lenguaje interpretado por el motor de Javascript, propietario de cada navegador. Esto no ser que los navegadores interpretan el código Javascript de distinta forma, es decir, el mismo código no obtiene los mismos resultados en distintos clientes.

Este hecho ha empujado la creación de bibliotecas Javascript, que son capas de abstracción de las operaciones básicas de este lenguaje. Estas librerías proporcionan un API para código Javascript que se utiliza para realizar operaciones complejas que, de ser implementadas manualmente, conllevarían evitar las diferencias entre navegadores y particularidades en el uso de eventos y otros mecanismos propios del lenguaje.

Las librerías pueden ser gratuitas o de pago y suelen ser orientadas a objetos. Javascript no es un lenguaje totalmente orientado a objetos pero permite simular algunos de los mecanismos de encapsulación, herencia y polimorfismo que definen este paradigma.

Por todo lo anterior parece razonable estudiar si estas librerías son útiles para el propósito de este trabajo, cómo funcionan, qué aportan, qué se puede hacer con ellas y cómo encajarían el modelado UML con ellas.

En el mercado existe una enorme cantidad de librerías y es imposible estudiarlas todas. En este trabajo se ha elegido *ExtJS* [17] de Sencha Labs, porque resulta una librería muy completa, que extiende la funcionalidad de las más famosas: *Prototype* [18], *JQuery* [19] y *YUI* [20].

Sencha Labs [21] es una fundación situada en California cuya misión es crear frameworks y herramientas que faciliten las labores del desarrollo RIA, evitando los detalles de la eficiencia e independencia del navegador a los desarrolladores web. Los productos de Sencha son open source y disponen de licencia comercial y no comercial.

Desde abril de 2011 proporciona la cuarta versión de su librería ExtJS, el producto que nos interesa. ExtJS salió a la luz en abril de 2007 con su primera versión estable lo cual hace de esta las más joven si la comparamos con otras como Dojo Toolkit (2004) [22], JQuery (2006), Prototype (2005), YUI (2005) o Script.aculo.us (2005) [23]. La librería ExtJS proporciona a los desarrolladores:

1. Gran cantidad de *Widgets*:
    - Tablas.
    - Gráficas.
    - Pestañas.
    - Ventanas.
    - Arboles.
    - Layouts.
    - Barras de herramientas, menús y menús contextuales.
    - Listas desplegables.
    - Formularios.
    - Calendarios.
    - Etc.
2. APIs de utilidades:
    - Accesibilidad.
    - *Drag and Drop.*

- Modelo Vista Controlador.
- AJAX
- Integración con otras APIs como las de Google.

3. Documentación.
4. Independencia para los navegadores:
    - Safari a partir de la versión 3.
    - Mozilla Firefox a partir de la versión 3.6, tanto para PC como para Mac
    - Internet Explorer a partir de la versión 6.
    - Google Chrome a partir de la versión 6.
    - Opera a partir de la versión 10.5

Un ejemplo real de esta tecnología es Kohive [24], proyecto online muy interesante que aún se encuentra en su versión Beta. Se trata de un escritorio online al que los usuarios pueden acceder desde cualquier lugar. De esta forma los usuarios tienen siempre disponible un espacio de trabajo en el mismo estado en el quedó en su último uso. También ofrece la posibilidad de crear y compartir con otros usuarios subespacios de trabajo donde puedan trabajar en equipo en tiempo real y una galería de aplicaciones propietarias para su uso en el escritorio. En sus próximos objetivos está la liberación de un API para el desarrollo de estas aplicaciones.

Pero, ¿cómo es el código Javascript que implementa alguno de estos widgets? Las librerías Javascript suelen apoyarse en las características basadas en objetos del lenguaje y facilitan al programador la tarea de mantener este paradigma de programación. Se puede ver que el siguiente código [25] se asemeja a la programación orientada a objetos:

```
//Importar objetos
Ext.require([
    'Ext.grid.*',
    'Ext.data.*',
    'Ext.panel.*',
    'Ext.layout.container.Border'
]);
//Evento de inicialización
Ext.Loader.onReady(function() {
    Ext.define('Book',{
        extend: 'Ext.data.Model',
        fields: [
            {name: 'Author', mapping: 'ItemAttributes > Author'},
            'Title',
            'Manufacturer',
            'ProductGroup',
            'DetailPageURL'
        ]
```

```
    });

      /*
      Definición de el objeto BookStore, que almacena registros de libros:
      Autor, Editorial, Grupo de productos y página con detalles sobre la
      publicación
      */
        Ext.define('App.BookStore', {extend: 'Ext.data.Store',
        constructor: function(config) {
            config = config || {};
            config.model = 'Book';
            config.proxy = {
                type: 'ajax',
                url: 'sheldon.xml',
                reader: Ext.create('Ext.data.reader.Xml', {
                    record: 'Item',
                    id: 'ASIN',
                    totalRecords: '@total'
                })
            };
            App.BookStore.superclass.constructor.call(this, config);
        }
    });


        /*Tabla en la que se muestra la información anterior*/
    Ext.define('App.BookGrid', {
        extend: 'Ext.grid.Panel', alias: 'widget.bookgrid',

        // override
        initComponent : function() {
            this.columns = [
                {text: "Author", width: 120, dataIndex: 'Author', sortable:
true},
                {text: "Title", flex: 1, dataIndex: 'Title', sortable: true},
                {text: "Manufacturer", width: 115, dataIndex: 'Manufacturer',
sortable: true},
                {text: "Product Group", width: 100, dataIndex:
'ProductGroup', sortable: true}
            ];

            this.store = new App.BookStore({
                storeId: 'gridBookStore',
                url: 'sheldon.xml'
            });
            App.BookGrid.superclass.initComponent.call(this);
        }
    });


    /*Panel que muestra la información específica de un libro al pulsar sobre
la línea que lo contiene*/
    Ext.define('App.BookDetail', {
        extend: 'Ext.Panel',
        alias: 'widget.bookdetail',
        tplMarkup: [
```

```
            'Title: <a href="{DetailPageURL}"
target="_blank">{Title}</a><br/>',
            'Author: {Author}<br/>',
            'Manufacturer: {Manufacturer}<br/>',
            'Product Group: {ProductGroup}<br/>'
        ],
        startingMarkup: 'Please select a book to see additional details',

        bodyPadding: 7,
        initComponent: function() {
            this.tpl = Ext.create('Ext.Template', this.tplMarkup);
            this.html = this.startingMarkup;

            this.bodyStyle = {
                background: '#ffffff'
            };

            App.BookDetail.superclass.initComponent.call(this);
        },

        updateDetail: function(data) {
            this.tpl.overwrite(this.body, data);
        }
    });


        /*Panel que incluye tanto la table que muestra los libros como el
panel que muestra su detalle. Podría considerarse el Layout principal*/
    Ext.define('App.BookMasterDetail', {
        extend: 'Ext.Panel',
        alias: 'widget.bookmasterdetail',

        frame: true,
        title: 'Book List',
        width: 540,
        height: 400,
        layout: 'border',

        initComponent: function() {
            this.items = [{
                xtype: 'bookgrid',
                itemId: 'gridPanel',
                region: 'north',
                height: 210,
                split: true
            },{
                xtype: 'bookdetail',
                itemId: 'detailPanel',
                region: 'center'
            }];

            App.BookMasterDetail.superclass.initComponent.call(this);
        },

        initEvents: function() {
```

```
            App.BookMasterDetail.superclass.initEvents.call(this);

            var bookGridSm =
this.getComponent('gridPanel').getSelectionModel();
            ('selectionchange', function(sm, rs) {
            if (rs.length) {
                var detailPanel = Ext.getCmp('detailPanel');
                bookTpl.overwrite(detailPanel.body, rs[0].data);
            }
        })
            bookGridSm.on('selectionchange', this.onRowSelect, this);
        },
        onRowSelect: function(sm, rs) {
            if (rs.length) {
                var detailPanel = this.getComponent('detailPanel');
                detailPanel.updateDetail(rs[0].data);
            }

        }
    });
}, false);


//Instanciar todo lo anterior a un div HTML con nombre 'binding-example'
Ext.onReady(function() {
    var bookApp = new App.BookMasterDetail({
        renderTo: 'binding-example'
    });
    Ext.data.StoreManager.get('gridBookStore').load();
});
```

Este código genera la siguiente GUI, que quedaría dentro de una página HTML como en la figura 2.1.

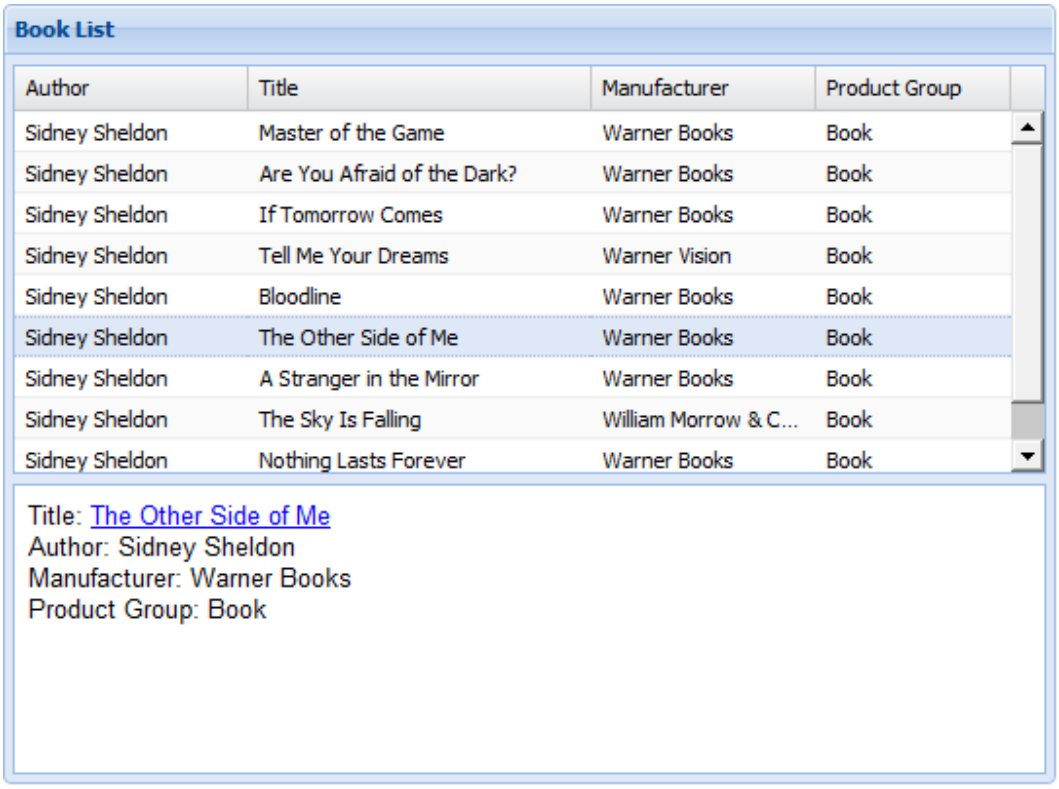


Figura 2.1 Ejemplo de pantalla ExtJS

2.2.2 Adobe Flex

Ya se ha expuesto anteriormente que existen alternativas al estándar `XMLHttpRequest` y una de ellas es la utilización de plugins para los navegadores web que extiendan la funcionalidad de éstos. Los plugins pueden utilizar sus propios mecanismos de compilación, interpretación y/o ejecución así como protocolos de comunicación para conseguir que ciertos objetos incrustados en el código HTML puedan ejecutarse de manera independiente al resto de la página y, entre otras posibilidades, pueda realizar llamadas síncronas al servidor.

Esta es la filosofía en la que se integra el funcionamiento de Flex una de las tecnologías objeto de estudio de este trabajo y un importante rival para AJAX. Flex apareció en marzo de 2004 como la apuesta de *Macromedia* (Posteriormente adquirida por Adobe) por llevar su programa estandarte, Flash, un paso adelante y convertirlo en una herramienta común en el desarrollo de RIA. Desde noviembre de 2011 está disponible la versión 4.6 de esta tecnología.

El uso de plugins para el navegador implica la descarga e instalación de software desarrollado por terceros aparte del propio navegador. Esto puede resultar un hándicap para un nuevo proveedor pero no para Adobe. Hoy en día el uso de Flash se ha extendido a lo largo y ancho de la Web: miles de anuncios, de webs, de animaciones, de aplicaciones en la nube y aplicaciones masivas como *Youtube* utilizan esta tecnología y, cuando un usuario sin este plugin llega a cualquiera de los puntos anteriores es avisado de que tiene instalar dicho plugin y se le facilita el enlace al software. Si bien esto también es posible para otros plugins, Adobe ya es una firma conocida gracias a otro software como Photoshop o Acrobat y puede generar mayor confianza que terceros desconocidos, lo cual representa una ventaja real puesto que los usuarios no siempre confían en la instalación de software en sus computadores. Pero ¿Por qué se está hablando indistintamente de Flash y de Flex? La razón es muy sencilla: Flex es un subconjunto de Flash. La tecnología Flex está enfocada al desarrollo RIA así como Flash está enfocado a la animación. Siendo desarrolladores web y diseñadores roles muy distinto, es muy difícil conseguir que la filosofía de Flash encaje en un desarrollador. Por ello Macromedia creó un conjunto de objetos, herramientas y utilidades basados en Flash con un API y una metodología de desarrollo más cercana a la que prefieren los desarrolladores.

Para desarrollar un componente Flex que se integre en una web se dispone de una extensa galería de componentes y dos lenguajes para reutilizar y extender su funcionalidad así como crear nuevos elementos desde cero: *MXML* [26] y *ActionScript* [27].

MXML es un lenguaje muy útil para crear composiciones de objetos: con el podemos dar forma a un nuevo componente utilizando layouts y otros componentes (tanto predefinidos como creados por el usuario). De esta manera se pueden crear escenarios muy diversos: desde mapas hasta formularios pasando por gráficos estadísticos y juegos online. MXML proporciona una manera estática de uso de componentes y sólo accede a las propiedades de estos para asignarles el valor de funciones que manejen sus eventos.

Estas funciones se implementan con ActionScript. Este es un lenguaje de desarrollo orientado a objetos con el que se pueden crear objetos de manera dinámica, manipular sus propiedades, realizar llamadas asíncronas a servidores, usar funciones del motor de simulación de física, así como otras funcionalidades diversas. Con él se accede a toda la potencia de Flex obviando los detalles que se dejan para MXML.

Además de las herramientas y componentes nativos que se incluyen en el SDK de Flex existen en internet otros componentes open source que pueden ser utilizados manteniendo la licencia. Algunos de los objetos que podemos encontrar en la distribución de Adobe Flex son:

1. Controles:
    a. Tablas
    b. Botones
    c. Barras de botones
    d. CheckBoxes
    e. Seleccionador de colores
    f. Listas desplegables
    g. Seleccionador de fechas
    h. Marcos de imágenes.
    i. Listas
    j. Menús
    k. Cajas de texto
    l. Pestañas
    m. Reproductor de video
    n. Etc.
2. Layouts:
    a. Bordes.
    b. Paneles.
    c. Ventanas.
    d. Agrupaciones verticales y horizontales.
    e. Agrupaciones en mosaico.
    f. Etc.
3. Navegación
    a. Paneles de acordeón.
    b. Navegación por pestañas.
    c. Pilas de paneles.
4. Diagramas:
    a. Barras.

b. Áreas.
c. Burbujas.
d. Columnas.
e. Líneas
f. De tarta.

5. Efectos:
    a. Transformaciones
    b. Desvanecimiento
    c. Crossfade
    d. Movimientos
    e. Rotaciones
    f. Cambio de tamaño.

Un ejemplo de uso de esta tecnología es la aplicación “Creative Studio” de HP [28]**.** En esta página se organizan elementos mediante paneles, pestañas y listas desplegables. Además utiliza animaciones y automatizaciones para mostrar la información a los usuarios.

Para poder aclarar cómo son las GUI realizadas con Flex así como el código MXML y ActionScript puede observarse la figura 2.2, donde se muestra el widget AdvancedDataGrid. Se trata de una tabla donde no sólo se puede mostrar información, sino también ordenarla por cualquier columna, así como modificar su visualización de tal forma que se pueda variar el orden de las columnas mostradas.

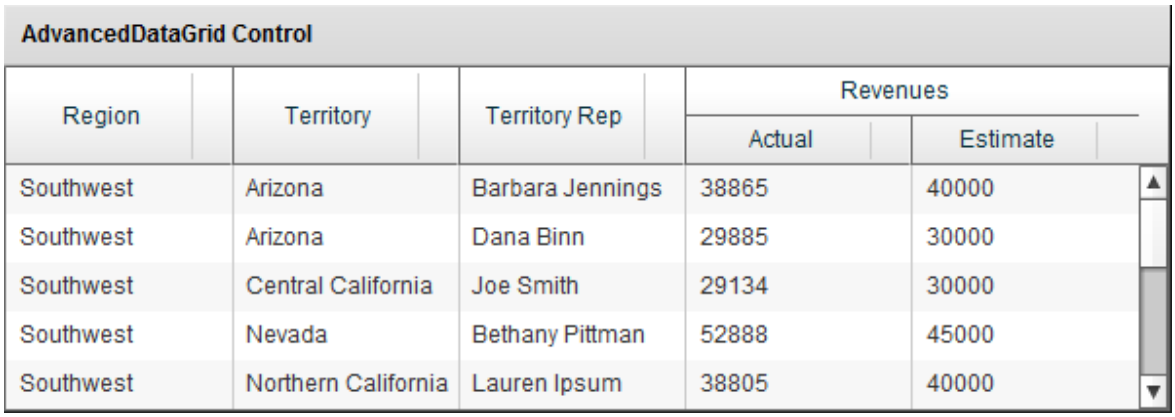


Figura 2.2 Ejemplo de AdvacedDataGrid, de Flex

El código fuente que crea este objeto y lo puebla con los datos que se pueden observar es el que se muestra a continuación. Como se puede observar es un archivo XML con una etiqueta <fx:Script> donde se incorpora el código ActionScript mencionado anteriormente. En este ejemplo [29] concreto sólo se utiliza para incorporar datos a la tabla:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               skinClass="TDFGradientBackgroundSkin"
               viewSourceURL="srcview/index.html">

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]private var dpFlat:ArrayCollection = new
ArrayCollection([
                {Region:"Southwest", Territory:"Arizona",
                    Territory_Rep:"Barbara Jennings", Actual:38865,
Estimate:40000},
                {Region:"Southwest", Territory:"Arizona",
                    Territory_Rep:"Dana Binn", Actual:29885, Estimate:30000},
                {Region:"Southwest", Territory:"Central California",
                    Territory_Rep:"Joe Smith", Actual:29134, Estimate:30000},
                {Region:"Southwest", Territory:"Nevada",
                    Territory_Rep:"Bethany Pittman", Actual:52888,
Estimate:45000},
                {Region:"Southwest", Territory:"Northern California",
                    Territory_Rep:"Lauren Ipsum", Actual:38805,
Estimate:40000},
                {Region:"Southwest", Territory:"Northern California",
                    Territory_Rep:"T.R. Smith", Actual:55498,
Estimate:40000},
                {Region:"Southwest", Territory:"Southern California",
                    Territory_Rep:"Alice Treu", Actual:44985,
Estimate:45000},
                {Region:"Southwest", Territory:"Southern California",
                    Territory_Rep:"Jane Grove", Actual:44913,
Estimate:45000},
                {Region:"NorthEast", Territory:"New York",
                    Territory_Rep:"Jose Rodriguez", Actual:26992,
Estimate:30000},
                {Region:"NorthEast", Territory:"New York",
                    Territory_Rep:"lisa Sims", Actual:47885, Estimate:50000},
                {Region:"NorthEast", Territory:"Massachusetts",
                    Territory_Rep:"kelly o'connell", Actual:172911,
Estimate:20000},
                {Region:"NorthEast", Territory:"Pennsylvania",
                    Territory_Rep:"John Barnes", Actual:32105,
Estimate:30000},
                {Region:"MidWest", Territory:"Illinois",
```

```
                    Territory_Rep:"Seth Brown", Actual:42511, Estimate:40000}
            ]);
        ]]>
    </fx:Script>

    <s:layout>
        <s:HorizontalLayout verticalAlign="middle" horizontalAlign="center"
/>
    </s:layout>

    <s:Panel title="AdvancedDataGrid Control"
             color="0x000000"
             borderAlpha="0.15"
             width="600">

        <mx:AdvancedDataGrid id="myADG"
                             width="100%" height="100%"
                             color="0x323232"
                             dataProvider="{dpFlat}">

            <mx:groupedColumns>

                <mx:AdvancedDataGridColumn dataField="Region" />
                <mx:AdvancedDataGridColumn dataField="Territory" />
                <mx:AdvancedDataGridColumn dataField="Territory_Rep"
                                           headerText="Territory Rep"/>
                <mx:AdvancedDataGridColumnGroup headerText="Revenues">
                    <mx:AdvancedDataGridColumn dataField="Actual"/>
                    <mx:AdvancedDataGridColumn dataField="Estimate"/>
                </mx:AdvancedDataGridColumnGroup>

            </mx:groupedColumns>

        </mx:AdvancedDataGrid>

    </s:Panel>

</s:Application>
```

2.2.3 Google Web Toolkit

La primera tecnología presentada en esta sección hablaba de métodos y utilidades que servían para programar aplicaciones web a un nivel mayor de abstracción. A este nivel se podían obviar detalles de diferencias entre navegadores o hacer más sencilla la gestión de primitivas o el uso del API `XMLHttpRequest`. Con ello se gana enfoque en el objetivo del desarrollo al no tener que preocuparse por detalles tecnológicos.

Por si esto fuera poco, en mayo de 2006, Google anunció el lanzamiento de la primera Release Candidate de Google Web Toolkit, GWT [30], que representa un nivel superior de

abstracción al propuesto en el párrafo anterior. Desde septiembre de 2011 se encuentra en la versión 2.4 ¿Cómo puede simplificarse aún más? Con GWT el desarrollador puede obviar la totalidad de los detalles de implementación de código Javascript y HTML de una página web puesto que esta tecnología permite desarrollar la interfaz web en código Java, al estilo de una interfaz de aplicación de escritorio Swing.

La idea de GWT es aumentar el enfoque del desarrollador de interfaces en su propósito y para ello parece lógico pensar que reducir el número de lenguajes es una buena idea. Si tenemos que utilizar un lenguaje para el desarrollo en el lado servidor de una aplicación web sería más fácil desarrollar la interfaz en ese mismo lenguaje. Podrían compartirse los objetos desde la capa de integración hasta la de vista sin pasar por complejas transformaciones a JSON o XML, se simplificarían las llamadas a métodos entre las distintas capas de la aplicación (por ejemplo para la implementación de patrones MVC y Observer), se podría modelar UML con la misma facilidad para toda la aplicación, no se requerirían expertos en Javascript.

Pero, ¿cómo es posible realizar esto? GWT utiliza un compilador que traduce el código Java a Javascript y HTML. Para ello, proporciona la GWT Class Library y un subconjunto del JRE de java que es susceptible de ser transformado a Javascript y HTML por motivos de limitaciones del lenguaje y de eficiencia. Todo este conjunto es utilizado por el compilador para la traducción. Si el compilador encuentra clases que no puede traducir avisará del error. De la misma forma, las clases de GWT Class Library no deben ser compiladas por el compilador Java ya que su finalidad última es la transformación a Javascript y HTML. Así mismo pueden desarrollarse clases que se compartan entre la vista y el resto de la aplicación. Estas clases podrían ser las que modelan los objetos que viajan desde la capa de integración a la de presentación y viceversa y deberían cumplir con las reglas del compilador GWT: no utilizar internamente tipos que no estén permitidos por el compilador, es decir, que no pertenezcan al subconjunto permitido del JRE.

GWT viene preparado para compilar clases Java y transformarlas en un código legible para todos los navegadores soportados. Para cada uno de ellos genera un archivo JS distinto. Es decisión del desarrollador si quiere que no se generen los archivos JS para algún cliente en concreto. Además, si se utilizan las capacidades multilenguaje de GWT se generará un archivo por cada par idioma-navegador. A estos pares se les denomina permutación en la nomenclatura de GWT.

El código generado por GWT funciona tal y como se espera y además es eficiente: el desarrollador puede concentrarse en la funcionalidad y no en la corrección del código. Puede mejorarse aún más la eficiencia con utilizando la ofuscación, un método utilizado de forma predeterminada por el compilador para minimizar la legibilidad y tamaño del JS, haciendo más rápida su descarga y más difícil su ingeniería inversa [31].

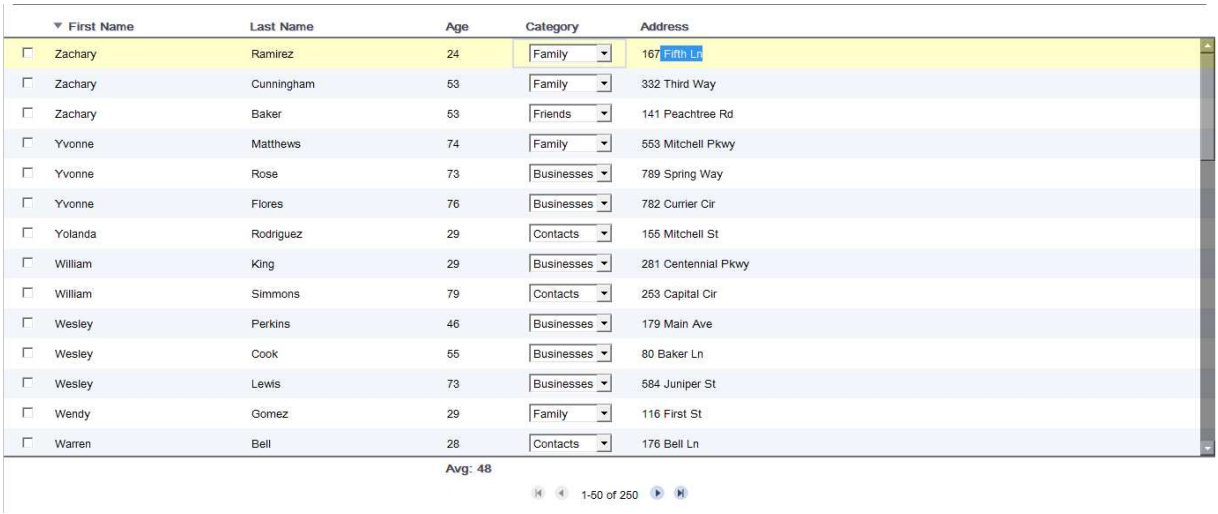
Además, GWT puede utilizarse para generar interfaces web en Java mientras que la tecnología del lado servidor puede ser cualquier otra como .NET o PHP. Esto es debido a que el código final generado es Javascript, que es capaz de comunicarse igualmente con casi todas las tecnologías de lado servidor mediante XML, JSON, texto plano, etc. De esta forma puede utilizarse GWT para generar una interfaz web que acceda a servicios publicados por un servidor que no sea Java. La ventaja de utilizar Java como tecnología en el servidor, es que las llamadas entre los métodos de las clases de distintas capas son transparentes para el desarrollador. No es necesaria la implementación implícita de llamadas asíncronas de ningún tipo ya que se encapsulan haciendo que las clases receptoras de llamadas desde la capa de presentación implementen una interfaz determinada, diseñada por Google para tal efecto.

Las principales ventajas de GWT son el enfoque orientado a objetos familiar a los desarrolladores y el consecuente uso de patrones de diseño y UML. La fácil reutilización de código, la depuración de código Java en eclipse en lugar de depuración en el navegador, la automatización de las pruebas mediante tests JUnit que simularían fácilmente la interacción de personas con la aplicación, los mecanismos de internacionalización, la ofuscación, la facilidad para gestionar el historial web, el aumento del rendimiento gracias al enfoque en un solo lenguaje de programación y la posibilidad de extender la funcionalidad de GWT con código Javascript en caso necesario.

No obstante también cuenta con inconvenientes: no todos los tipos de JRE son aceptados, el enfoque de desarrollo es inusual y los desarrolladores no confían en sus posibilidades, es difícil de utilizar sobre proyectos ya existentes, los widgets proporcionados por Google son muy sencillos y a veces es necesario extenderlos o utilizar librerías de widgets de terceros para el desarrollo de aplicaciones y que al utilizarse una sola página en la ejecución el enfoque de la seguridad es distinto.

Un ejemplo de éxito de esta tecnología podría ser cualquiera de la aplicaciones de Google: *Gmail, Google Docs o Google Adwords y Adsense*. Pero a continuación se muestra un

ejemplo de un widget de GWT del estilo de los ejemplos anteriores: una tabla que podría utilizarse en multitud de ocasiones en aplicaciones reales. La tabla de la figura 2.3 muestra una cabecera fija y una gran cantidad de datos a los que se accede mediante un scroll. Además, está paginada y es posible ordenar por columnas así como reordenar las columnas.



First Name	Last Name	Age	Category	Address
Zachary	Ramirez	24	Family	167 Fifth Ln
Zachary	Cunningham	53	Family	332 Third Way
Zachary	Baker	53	Friends	141 Peachtree Rd
Yvonne	Matthews	74	Family	553 Mitchell Pkwy
Yvonne	Rose	73	Businesses	789 Spring Way
Yvonne	Flores	76	Businesses	782 Currier Cir
Yolanda	Rodriguez	29	Contacts	155 Mitchell St
William	King	29	Businesses	281 Centennial Pkwy
William	Simmons	79	Contacts	253 Capital Cir
Wesley	Perkins	46	Businesses	179 Main Ave
Wesley	Cook	55	Businesses	80 Baker Ln
Wesley	Lewis	73	Businesses	584 Juniper St
Wendy	Gomez	29	Family	116 First St
Warren	Bell	28	Contacts	176 Bell Ln

Avg: 48

1-50 of 250

Figura 2.3 ejemplo de widget GWT

El código [32] que la implementa es totalmente orientado a objetos ya que se trata de código Java. Se omiten partes de código referentes a añadir las columnas de apellido, edad y dirección por su gran similitud con la columna para el nombre:

```
      //Objeto Datagrid parametrizado con el tipo de datos que va a contener
      @UiField(provided = true)
      DataGrid<ContactInfo> dataGrid;

      //Paginacion
      @UiField(provided = true)
      SimplePager pager;

      @Override
      public Widget onInitialize() {
          dataGrid = new
DataGrid<ContactInfo>(ContactDatabase.ContactInfo.KEY_PROVIDER);
          dataGrid.setWidth("100%");

          dataGrid.setEmptyTableWidget(new
Label(constants.cwDataGridEmpty()));

          ListHandler<ContactInfo> sortHandler =
```

```
            new
      ListHandler<ContactInfo>(ContactDatabase.get().getDataProvider().getLis
      t());
          dataGrid.addColumnSortHandler(sortHandler);

            //Configurar paginacion
            SimplePager.Resources pagerResources =
      GWT.create(SimplePager.Resources.class);
          pager = new SimplePager(TextLocation.CENTER, pagerResources, false,
0, true);
          pager.setDisplay(dataGrid);

          // Configurar cómo se selecciona en la tabla
          final SelectionModel<ContactInfo> selectionModel =
              new
MultiSelectionModel<ContactInfo>(ContactDatabase.ContactInfo.KEY_PROVIDER);
          dataGrid.setSelectionModel(selectionModel,
DefaultSelectionEventManager
              .<ContactInfo> createCheckboxManager());

          initTableColumns(selectionModel, sortHandler);
          ContactDatabase.get().addDataDisplay(dataGrid);

          Binder uiBinder = GWT.create(Binder.class);
          return uiBinder.createAndBindUi(this);
        }

        //Añadir columnas a la tabla
        private void initTableColumns(final SelectionModel<ContactInfo>
selectionModel,
            ListHandler<ContactInfo> sortHandler) {
          //Columna con checkboxes
          Column<ContactInfo, Boolean> checkColumn =
              new Column<ContactInfo, Boolean>(new CheckboxCell(true, false))
{
                @Override
                public Boolean getValue(ContactInfo object) {
                  // Get the value from the selection model.
                  return selectionModel.isSelected(object);
                }
              };
          dataGrid.addColumn(checkColumn,
SafeHtmlUtils.fromSafeConstant("<br/>"));
          dataGrid.setColumnWidth(checkColumn, 40, Unit.PX);

          // Nombre
          Column<ContactInfo, String> firstNameColumn =
              new Column<ContactInfo, String>(new EditTextCell()) {
                @Override
                public String getValue(ContactInfo object) {
                  return object.getFirstName();
                }
              };
          firstNameColumn.setSortable(true);
          sortHandler.setComparator(firstNameColumn, new
Comparator<ContactInfo>() {
```

```
            public int compare(ContactInfo o1, ContactInfo o2) {
              return o1.getFirstName().compareTo(o2.getFirstName());
            }
          });
          dataGrid.addColumn(firstNameColumn,
constants.cwDataGridColumnFirstName());
          firstNameColumn.setFieldUpdater(new FieldUpdater<ContactInfo,
String>() {
            public void update(int index, ContactInfo object, String value) {
              // Called when the user changes the value.
              object.setFirstName(value);
              ContactDatabase.get().refreshDisplays();
            }
          });
          dataGrid.setColumnWidth(firstNameColumn, 20, Unit.PCT);

          // ComboBox de categoría
          final Category[] categories =
ContactDatabase.get().queryCategories();
          List<String> categoryNames = new ArrayList<String>();
          for (Category category : categories) {
            categoryNames.add(category.getDisplayName());
          }
          SelectionCell categoryCell = new SelectionCell(categoryNames);
          Column<ContactInfo, String> categoryColumn = new
Column<ContactInfo, String>(categoryCell) {
            @Override
            public String getValue(ContactInfo object) {
              return object.getCategory().getDisplayName();
            }
          };
          dataGrid.addColumn(categoryColumn,
constants.cwDataGridColumnCategory());
          categoryColumn.setFieldUpdater(new FieldUpdater<ContactInfo,
String>() {
            public void update(int index, ContactInfo object, String value) {
              for (Category category : categories) {
                if (category.getDisplayName().equals(value)) {
                  object.setCategory(category);
                }
              }
              ContactDatabase.get().refreshDisplays();
            }
          });
          dataGrid.setColumnWidth(categoryColumn, 130, Unit.PX);
```

2.2.4 HTML5

HTML 5 [33] es el futuro estándar para el desarrollo de interfaces web. La quinta revisión del lenguaje de marcado HTML se halla aún en estado de desarrollo pero el borrador actual está muy avanzado. Tanto es así que se pueden encontrar ya en la red multitud de ejemplos de sus nuevas característica y no son pocos los esfuerzos que están haciendo para

conseguir que se adopte lo antes posible: Google [34], Mozilla [35], Microsoft [36] y Apple [37] tienen sus propios recursos para difundirlo y formar a los desarrolladores.

La nueva versión del lenguaje trae consigo mejoras sustanciales en muchos aspectos que, a causa del avance de otras tecnologías y las necesidades de los usuarios, habían quedado obsoletas, facilitaban malas prácticas o no ayudaban a la accesibilidad de la web. Además, se están incorporando nuevas funcionalidades y mecanismos de extensión para evitar la obsolescencia temprana a la que puede verse sometida, a tenor de la rápida evolución de la web.

Unas novedades vienen como nuevos elementos estructurales o semánticos. Estos elementos ayudan a dividir una página en secciones o bien aportan contenido semántico. Algunas de estas etiquetas son:

- <header> Especifica una introducción para el documento.
- <footer> Indica un pie de página, asignable a una sección.
- <section> Define una sección en un documento.
- <nav> Define enlaces de navegación.
- <details> Describe una zona aaaa por el usuario.
- <time> indica una fecha u hora.
- <mark> indica un texto resaltado.

Otras son los más publicitados elementos para la incrustación de audio y video <audio> y <video> así como el elemento <canvas> que permite dibujar gráficos dinámicamente con JavaScript [38]. Estas etiquetas traen sus correspondientes propiedades y métodos que posibilitan la interacción con la reproducción del video, audio o el propio lienzo. No obstante también se han eliminado otras etiquetas por su falta de uso o uso indebido. Algunas de ellas, como <applet>, <frame> y <frameset> han sido mencionadas antes en este trabajo.

Muchos navegadores ya soportan las novedades que trae HTML 5 con respecto a la funcionalidad "Drag & Drop". Esta funcionalidad se conseguía hasta ahora únicamente mediante bibliotecas JavaScript y consiste en la posibilidad de arrastrar ciertos elementos de una página de un lugar a otro. Con HTML 5, todos Los elementos pueden ser arrastrados y soltados fácilmente ya que proporciona en el estándar un API que puede ser utilizada de manera muy sencilla desde JavaScript.

También se contará con soporte nativo a los gráficos SVG, que son gráficos vectoriales que se pueden crear y editar con cualquier editor de textos, pueden escalarse sin degradarse su

calidad, así como imprimirse en cualquier resolución. Las cookies podrán ser reemplazadas por el Web Storage, que es un almacenamiento en el navegador, más rápido, seguro (un sitio web solo puede acceder a datos almacenados por ella) y no afecta al rendimiento de los sitios web.

Existen otras dos mejoras que afectan seriamente a la forma en la que estamos acostumbrados a ver la web: los web workers y los Server-Sent Events. Los primeros consisten en hilos de ejecución paralela de Javascript, lo cual permite ejecutar código Javascript en segundo plano sin bloquear la página. Los segundos son eventos invocados por el servidor. Actualmente sólo pueden enviarse eventos del cliente al servidor, y las actualizaciones constantes que se ven en sitios como Twitter o Facebook se consiguen mediante llamadas asíncronas programadas cada cierta cantidad de tiempo, en las que se piden actualizaciones. Con los eventos disparados desde el servidor, el propio servidor actualizaría la página sin tener que estar recibiendo peticiones constantemente.

HTML 5 se combina con la nueva versión de CSS [39] dándole una nueva dimensión a las posibilidades actuales. A día de hoy es impensable poder realizar una web con distintas transiciones entre elementos (bien sean textos, fotografías, etc.) sin utilizar gran cantidad de código Javascript o incluso Flash. Tampoco es posible utilizar textos de gran tamaño, girados en cierta medida o con algún grado transparencia a no ser que se utilicen imágenes.

Pero a pesar de todos estos avances, para hacer RIA hay que seguir utilizando frameworks o bibliotecas de Javascript ya que HTML es un lenguaje que sigue siendo insuficiente. No obstante, estos frameworks evolucionarán debido a las nuevas APIs y funcionalidad del estándar. A continuación se muestra una lista de tecnologías que apoyan HTML 5 pero no pertenecen a éste estándar.

- WebGL
- XMLHttpRequest
- Geolocalización
- CSS3
- ECMAScript5
- XBL2
- Web Sockets

2.2.5 JSF

Java Server Faces, JSF, es un framework para el desarrollo de componentes de interfaz de usuario de lado servidor de aplicaciones web Java [40]. El objetivo de JSF es proporcionar un marco de trabajo en el que se representen y manipulen los componentes de interfaz de usuario así como sus eventos en código Java y en el lado servidor. Dentro de este marco de trabajo también se puede realizar la validación de datos de formularios y su conversión entre tipos, definir el flujo de navegación entre páginas, dar soporte al desarrollo de interfaces multilenguaje y accesibles para usuarios con discapacidad. También se disponen de mecanismos para extender las posibilidades de estas funcionalidades.

La forma en la que se representa esta interfaz desarrollada en el lado servidor es mediante la utilización de un conjunto de etiquetas JSP *Java Server Faces*, [41]. Estas etiquetas se vinculan a los componentes desarrollados y sirven para el envío bidireccional de datos.

De esta forma, una aplicación JSF consta de:

1. Un conjunto de páginas JSP donde habrá un subconjunto que contenga etiquetas JSF.
2. Un conjunto de *beans* [42] de lado servidor que son los que definen las propiedades y funciones de los componentes de interfaz.
3. Un archivo de configuración de la aplicación donde se configuran estos beans y se definen reglas de navegación.
4. El descriptor de despliegue web.xml
5. De forma opcional podría haber objetos encargados de validar y convertir datos y manejadores de eventos.
6. También de forma opcional, un conjunto de etiquetas jsp personalizadas para los objetos anteriores.

Para entender más profundamente JSF es importante conocer el ciclo de vida de las páginas JSF. Distinguimos entre dos tipos de acceso a páginas JSF que definen dos ciclos de vida distintos: peticiones iniciales y postbacks. Las peticiones iniciales a una página JSF son aquellas en las que accedemos por primera vez a la aplicación y las postbacks aquellas que se realizan sobre páginas JSF directamente. Normalmente y en el primer caso, el servidor recibirá la petición HTTP y pasará por las fases de Restaurar Vista, Aplicar Peticiones, y Generar Respuesta. Así mismo y como norma general, en el segundo caso pasará por *Restaurar Vista, Aplicar*

Peticiones, Procesar Validaciones, Actualizar Valores del Modelo, Invocar Aplicación y *Generar Respuesta*.

1. *Restaurar Vista*: se construye la vista (árbol de clases) de la página, se enlazan a ella sus manejadores de eventos y validadores y se guarda en un almacén de contextos. En una petición inicial esta vista se crea vacía y se puebla con aquellos componentes que se referencian mediante etiquetas en la página. En el segundo caso, si es un postback, la vista ya existe y lo que se hace es recuperarla del almacén de contextos.
2. *Aplicar Petición*: en esta fase se obtienen los valores de los parámetros incluidos en la petición y se almacenan. Estos valores pueden ser susceptibles de ser convertidos y en caso de no superar la conversión se genera un mensaje de error que se encola para ser mostrado en la fase de Generar Respuesta. Si la finalidad de la petición es redirigir una página fuera del contexto JSF simplemente se redirige y termina el ciclo de vida.
3. *Procesar Validaciones*: Fase en la que se realizan las validaciones que se han configurado para los componentes de la interfaz. Como en el caso anterior, los errores de validación se encolan para ser mostrados en la última etapa del ciclo de vida.
4. *Actualizar Valores del Modelo*: Si los datos han sido validados satisfactoriamente en la etapa anterior, se actualizan los valores de los atributos de los beans del modelo.
5. *Invocar Aplicación*: En esta fase, JSF ejecuta el código de los manejadores de eventos.
6. *Generar respuesta*: Final del ciclo de vida de una petición JSF en la que se generan los componentes necesarios para la visualización de la página JSP si es necesario o los mensajes de error fruto del resto de pasos anterior.

La primera versión de JSF salió a la luz en marzo de 2004 y actualmente está disponible la versión 2.1. Desde JSF 2.0 el soporte a AJAX es nativo, estandarizado y no es necesario extender la funcionalidad de los componentes JSF con código Javascript. El uso de las etiquetas <h: ajax> indican a JSF que el componente realiza la llamada de forma asíncrona en lugar de síncrona.

The Vanguard Group Inc [43] es una firma estadounidense de inversores. Visitando su web se pude observar un ejemplo de utilización de tecnología JSF con llamadas asíncronas. No obstante, a continuación se muestra un ejemplo de código JSF [44] que se incluye en una página JSP para mostrar una tabla como las que se han mostrado a lo largo de este capítulo. La implementación de esta tabla en concreto es IceFaces, una de las múltiples distribuciones que cumplen con el estándar JSF.

```
<body>
<h2>Sortable dataTable Component</h2>
<ice:form>
    <ice:dataTable
            id="dataSortData"
            sortColumn="#{inventoryList.sortColumnName}"
            sortAscending="#{inventoryList.ascending}"
            value="#{inventoryList.carInventory}"
            var="item">

        <!-- Identificador -->
        <ice:column>
            <f:facet name="header">
                <ice:commandSortHeader

columnName="#{inventoryList.stockColumnName}"
                                arrow="true" >
                            <ice:outputText
value="#{inventoryList.stockColumnName}"/>
                </ice:commandSortHeader>
            </f:facet>
            <ice:outputText value="#{item.stock}"/>
        </ice:column>

        <!—Modelo  -->
        <ice:column>
            <f:facet name="header">
                <ice:commandSortHeader

columnName="#{inventoryList.modelColumnName}"
```

```
                                arrow="true" >
                            <ice:outputText
value="#{inventoryList.modelColumnName}"/>
                        </ice:commandSortHeader>
                    </f:facet>
                    <ice:outputText value="#{item.model}"/>
                </ice:column>

                <!-- Descripción  -->
                <ice:column>
                    <f:facet name="header">
                        <ice:commandSortHeader

columnName="#{inventoryList.descriptionColumnName}"
                                arrow="true" >
                            <ice:outputText
value="#{inventoryList.descriptionColumnName}"/>
                        </ice:commandSortHeader>
                    </f:facet>
                    <ice:outputText value="#{item.description}"/>
                </ice:column>

                <!—Lectura del sensor -->
                <ice:column>
                    <f:facet name="header">
                        <ice:commandSortHeader

columnName="#{inventoryList.odometerColumnName}"
                                arrow="true" >
                            <ice:outputText
value="#{inventoryList.odometerColumnName}"/>
                        </ice:commandSortHeader>
                    </f:facet>
                    <ice:outputText value="#{item.odometer}"/>
                </ice:column>

                <!-- Precio -->
                <ice:column>
```

```
                    <f:facet name="header">
                        <ice:commandSortHeader

columnName="#{inventoryList.priceColumnName}"
                                    arrow="true" >
                                <ice:outputText
value="#{inventoryList.priceColumnName}"/>
                        </ice:commandSortHeader>
                    </f:facet>
                    <ice:outputText value="#{item.price}"/>
                </ice:column>
            </ice:dataTable>
        </ice:form>
        </body>
```

Pero es importante también mostrar el código del “Backing Bean” que utiliza esta tabla para obtener tanto los datos como otra información necesaria como los criterios de ordenación (mediante los mecanismos de introspección):

```
    public abstract class SortableList {
        protected String sortColumnName;
        protected boolean ascending;
        protected String oldSort;
        protected boolean oldAscending;

        protected SortableList(String defaultSortColumn) {
            sortColumnName = defaultSortColumn;
            ascending = isDefaultAscending(defaultSortColumn);
            oldSort = sortColumnName;
            oldAscending = !ascending;
        }

        protected abstract void sort();

        protected abstract boolean isDefaultAscending(String sortColumn);

        public String getSortColumnName() {return sortColumnName; }
```

```
    public void setSortColumnName(String sortColumnName) {
        oldSort = this.sortColumnName;
        this.sortColumnName = sortColumnName;
    }

    public boolean isAscending() { return ascending;}

    public void setAscending(boolean ascending) {
        oldAscending = this.ascending;
        this.ascending = ascending;
    }
}
```

Capítulo 3 Diseño de aplicaciones RIA

El auge de las tecnologías RIA ha llevado al desarrollo de distintas aproximaciones de modelado para el diseño este tipo de aplicaciones. En este capítulo se analizan brevemente algunas de las principales notaciones que han incluido primitivas de modelado para aplicaciones RIA, así como algunos de los principales catálogos de patrones RIA.

Cabe destacar en este apartado la notación *UML Web Application Extension* (UML WAE) [7] definida por Jim Conallen. A pesar de que UML WAE es la notación que permite un uso de UML más cercano a la filosofía propuesta en el proceso unificado de desarrollo, y por tanto al código, en la actualidad no presenta ninguna extensión para aplicaciones RIA, por lo que no ha sido incluida en esta sección.

3.1 Notaciones de modelado

3.1.1 RUX-Model

RUX-Model [4] es una aproximación que parte de la situación en la que se encuentran muchas aplicaciones web cuya interfaz es más propia de la web 1.0 que de la 2.0. En este escenario se busca migrar estas aplicaciones a una interfaz más propia de RIA y ahí es donde esta metodología centra sus esfuerzos: un método de ingeniería del software con el que proporcionar una nueva interfaz a aplicaciones estructuradas en capas en las que se puedan utilizar las capas de negocio y de datos.

Utiliza su propia notación que está implementada en su propia herramienta de diseño y divide la interfaz en cuatro niveles:

- Conceptos y tareas: datos y lógica de negocio (heredados de la aplicación existente)
- Interfaz abstracta: una representación común a todos los dispositivos y plataformas de desarrollo de RIA, una abstracción a la tecnología de implementación.

- Interfaz concreta: es la realización de un modelo de interfaz abstracta. A su vez se divide en tres niveles de presentación:
    - Espacial: en la que se representa la distribución espacial de la interfaz junto al *Look and Feel* deseado.
    - Temporal: en la que se detallan los comportamientos que requieren sincronización temporal.
    - De interacción: enfocada en el comportamiento del usuario.
- Interfaz final: específica para una tecnología de desarrollo RIA.

Este modelo tiene tres fases de transformación:

- CR (Reglas de conexión), en la que se extrae toda la información relevante del modelo de datos y de la lógica de negocio para construir de forma automática una versión inicial de la interfaz abstracta. Las reglas de conexión establecen un proceso de asignación entre los elementos del modelo de la web y los componentes de interfaz abstractos, pueden ser refinadas según las necesidades de la aplicación.
- TR1 (Reglas de transformación 1), en la que RUX-Model proporciona un borrador de la interfaz concreta automáticamente con la que se establecen relaciones entre componentes abstractos y concretos que también pueden ser refinadas.
- TR2 (Reglas de transformación 2), es la fase final en la que el modelo RUX proporciona la interfaz final basándose en la tecnología deseada.

El modelo RUX se asienta sobre una biblioteca de componentes multinivel que define los componentes que existen en cada uno de los niveles de presentación:

- Almacena la especificación de cada componente.
- Especifica la transformación y relaciones entre las propiedades de los componentes entre distintos niveles
- Mantiene la jerarquía entre componentes en cada nivel, independientemente de las jerarquías que haya en otros niveles.

3.1.2 Extensión de WebML

La extensión a WebML de Bozzon y Comai [5] está motivada en la reducción de la distancia existente entre las metodologías de desarrollo web convencional y el paradigma de RIA, haciendo hincapié en sus características comunes.

WebML utiliza diagramas UML o Entidad-Relación para modelar aplicaciones web. Según esta aproximación, para un modelo de datos se pueden definir diferentes vistas por cada rol de usuario de usuario que interactúa con el sistema. Una vista de sitio no es más que un conjunto de páginas que pueden estar agrupadas en secciones, su contenido está atomizado, y depende de una entidad del modelo de datos. El contenido es publicado dependiendo de condiciones lógicas (selectores), ordenado según determinados criterios y vinculado con otros contenidos mediante enlaces. Finalmente, existen las operaciones, que son invocaciones a la lógica de negocio del servidor.

La extensión realizada sobre WebML afecta a varias dimensiones:

- Mientras que en la web tradicional la persistencia sólo se estudia en el lado del servidor, en RIA existe mucho contenido de lado cliente así como objetos residentes en memoria que pueden tener la misma visibilidad y duración e incluso pueden persistir.
- Esto conlleva nuevos tipos de relaciones que no se contemplan en las aplicaciones web tradicionales. Las relaciones que pueda haber entre objetos del lado cliente y las relaciones entre objetos de lado cliente y servidor se añaden a las relaciones entre objetos de lado servidor.
- Aparecen las páginas cliente. A diferencia de las páginas de servidor, el contenido y lógica de éstas puede ser controlado, al menos en parte, por el cliente. La presentación, generación del contenido y manejo de eventos ocurren en el cliente.
- Como consecuencia del punto anterior, aparecen tanto operaciones como unidades de contenido de cliente, una situación no prevista por WebML que sólo contempla las de lado servidor.

3.1.3 OOH4RIA

La aproximación dirigida por modelos OOH4RIA [6] propone un conjunto de modelos y transformaciones para la generación de RIA. Está basada en OOH [45] que define modelos de

dominio y navegación para generar el código de la parte servidor de una aplicación web para las operaciones CRUD (*Create, Read, Update, Delete*) y se apoya en una herramienta de desarrollo basada en Eclipse *Graphical Modeling Framework* [46].

El proceso de OOH4RIA comienza definiendo el modelo del dominio con sus entidades y relaciones. Posteriormente se desarrolla el modelo de servidor en el que se representa la navegación entre conceptos del dominio así como las reglas que controlan la visualización del contenido en la interfaz.

A continuación se realiza el modelo de presentación estructural de los componentes visuales mediante interfaz gráfica que establece relaciones entre estos componentes entre sí y elementos del modelo de navegación. Esto se completa con diagramas de estado para cada componente.

El último paso del proceso son las transformaciones M2T entre los modelos y el código de la aplicación. OOH4RIA-Tool utiliza GWT como lenguaje objetivo de esta transformación.

Uno de los aspectos más destacados de OOH4RIA es la presencia de una taxonomía que clasifica de manera abstracta los principales componentes visuales de una RIA. Dicha taxonomía puede ser reutilizada por aproximaciones que deseen aplicar un desarrollo dirigido por modelos para la construcción de aplicaciones RIA.

3.1.4 OOWS

En contraposición a las posturas que utilizan distintos modelos para el desarrollo dirigido por modelos de RIA, OOWS [47] propone un único modelo, el Modelo de Interacción para definir todos los requisitos de interacción, donde interacción es toda acción tomada entre un usuario y el software para llevar a cabo una tarea.

Una tarea está compuesta de por Patrones Abstractos de Interacción (AIPs) que se dividen en Patrones de Obtención de Información o de Ejecución de Servicios. Como indica su nombre, estos patrones son abstractos y no definen detalles de las tecnologías que los implementan no como los Patrones Concretos de Interacción (CIPs) que son especializaciones de AIPs.

Los CIPs posteriormente pueden ser transformados en código mediante transformaciones M2T. Este código se dividiría en:

- Implementación de interfaz RIA (ExtJS, GWT, etc…)

- Fachada con la que comunicarse con la lógica de negocio a través de servicios AJAX y REST con XML.

OOWS proporciona dos AIPs principales:

- La AIP de Población representa una obtención de datos del sistema, que en realidad define una vista sobre una representación del modelo del dominio.
- La AIP de servicio es la abstracción de un servicio y se define como una vista de los argumentos de un servicio. Se crean Argumentos de Entrada para insertar los valores correspondientes.

Pero también existen otras AIPs secundarias:

- AIP de filtrado: restringe la información devuelta por una AIP de Población.
- AIP de selección definida: con junto de valores asociados a un Argumento de Entrada AIP.

3.1.5 UWE4JSF

UWE4JSF es una herramienta CASE creada para realizar el desarrollo de aplicaciones web en el ámbito del *UML-based Web Engineering* (UWE). Se trata de un plugin de Eclipse enfocado en la generación automática de aplicaciones web que proporciona una aproximación de alto rendimiento, depurable y entendible para las personas que utiliza JSF como tecnología subyacente.

Para ello se ha realizado una extensión de UWE al que se le ha añadido un modelo concreto de presentación. Este sirve para modelar componentes concretos de la interfaz como clases UML estereotipadas cuyas instancias son configuraciones de componentes concretos.

Esta aproximación cuenta además con un catálogo de patrones que será brevemente analizado en la próxima sección.

3.2 Patrones de RIA

3.2.1 Introducción

La ingeniería del software insiste en la reutilización no sólo del código, sino también de las soluciones, más o menos abstractas, que ya existan para problemas o situaciones comunes que los ingenieros encuentran en la labor del desarrollo de software.

Cuando estas soluciones son indiscutiblemente efectivas y pueden reutilizarse en distintos escenarios, abstrayéndolas en mayor o menor medida, las denominamos patrones. Los patrones se organizan en catálogos atendiendo a distintos criterios y su origen se encuentra en los patrones de arquitectura civil, que son capturas de diseños arquitectónicos como descripciones arquetípicas y reutilizables que sirven de ayuda en el diseño de ciudades y edificios.

La principal misión de los patrones es “no reinventar la rueda”: evitar el esfuerzo por encontrar la solución a un problema existente.

En el desarrollo de RIA esto también puede ocurrir y en la literatura se pueden encontrar variadas situaciones y las correspondientes soluciones propuestas, es decir, catálogos de patrones de RIA. Estos patrones están más enfocados en la interacción entre los usuarios y la web que en los esquemas organizativos del propio sistema

3.2.2 Catálogo de Koch

En el estudio de los patrones para el desarrollo basado en modelos de RIA [1] se define un patrón de RIA como “una solución general reutilizable para un problema que ocurra con frecuencia en el diseño de RIA”. Un patrón de RIA “describe la interacción, operación y presentación de un widget RIA”.

Los patrones de RIA de Nora Koch están orientados a estar lo suficientemente bien modelados como para poder ser utilizados en el desarrollo de RIA de forma manual o automática siguiendo el enfoque del desarrollo dirigido por modelos.

Su aproximación consiste en la utilización de modelos de widgets RIA y utilizarlos a su vez dentro de otros modelos ya existentes. El diseñador de RIA necesitaría:

- Modelos de patrones.
- Lenguaje que represente los eventos del usuario y del sistema.
- Elementos de modelado que se integren con modelos de aplicaciones web existentes.

En este catálogo de patrones podemos encontrar, por ejemplo, el patrón Auto-Completar. Este patrón de RIA consiste en ayudar a los usuarios en la tarea de rellenar formularios. Está motivado por lo tedioso que es rellenar campos que podrían ser repetidos y para evitar que los usuarios tengan que escribir mucho y por lo tanto reducir errores tipográficos y tiempo de entrada de datos.

Consiste en predecir la entrada de datos del usuario de dos formas distintas:

- La primera es en la copia o inferencia del contenido de los campos completamente. Imaginemos por ejemplo el caso en el que un usuario adquiere un billete de avión y utiliza la opción de emitir una factura. En este caso, parece lógico que los datos de facturación sean los mismos que los de la persona que va a viajar.
- La segunda es la predicción del resto de la palabra o frase que se está escribiendo, mediante la utilización de distintos recursos como puedan ser bases de datos alimentadas con estadísticas de lo que han escrito otros usuarios. Un ejemplo claro de este patrón puede encontrarse en la barra de búsqueda web de Google.

Otro patrón referenciado en el estudio de Koch es el Refreso Periódico. Este patrón es cada vez utilizado más ampliamente en toda la web. Su finalidad es proporcionar a los usuarios información en tiempo real sobre cualquier contenido dinámico.

La solución es realizar peticiones periódicas al servidor para ver si ha habido alguna actualización de los datos que se estén observando y, en caso afirmativo, mostrar esa nueva actualización.

Un famoso ejemplo de este funcionamiento es la página principal de la aplicación de microblogging Twitter, donde se pueden ver en tiempo real las últimas actualizaciones de los usuarios sobre un tema en concreto.

Para reducir los esfuerzos de modelado y desarrollo se propone la extensión de UWE (*UML-based Web Engineering*) que es un método para el desarrollo sistemático de aplicaciones web dirigido por modelos. En UWE se separa el modelado del contenido, de la estructura de navegación, de los procesos de negocio y de las representaciones abstractas y concretas de una aplicación web. Como su propio nombre indica, utiliza un perfil UML para el modelado.

3.2.3 Catálogo de Governor, Hinchcliffe y Nickull

En su libro *Web 2.0 Architectures* [2], Governor, Hinchcliffe y Nickull exponen varios patrones a un alto nivel de abstracción relativos a la web 2.0, de los cuales hay algunos que encajan perfectamente como patrones de RIA.

Así, la Actualización Asíncrona de Partícula es un patrón que engloba a las distintas técnicas explicadas en la sección anterior de este trabajo. Parte del problema común de todas es

que la web tradicional era insuficiente para sustituir a las aplicaciones de escritorio debido a la imposibilidad de carga asíncrona de contenido y propone que los navegadores, de alguna u otra forma (anteriormente se ha hablado de la interfaz `XMLHttpRequest` y de los plugins de Flash, por ejemplo) sea capaz de recibir una petición específica que sirva para manipular parte del documento representado de forma asíncrona.

Actualización Asíncrona de Partícula responde a la necesidad de actualizar una parte concreta de una página de forma asíncrona, sin tener que actualizar la página entera con la cantidad de inconvenientes que ello supone y que se han explicado anteriormente en este trabajo. Una implementación de este patrón es la tecnología AJAX, que da solución a esa necesidad utilizando los recursos Javascript existentes para ello.

Otro patrón del libro y que cada día tiene una mayor aceptación es el patrón Remezcla (Mashup). Este patrón consiste en la utilización de distintos servicios ofrecidos por terceros en una misma página, es decir, mezcla de servicios para proporcionar una vista compuesta por ellos. Este patrón se da cuando, por ejemplo, en un blog se publican fotografías mediante el API de Flickr o cuando se inserta un mapa de Google Maps en una web.

Tanto los proveedores de servicios para todo tipo de usuarios como Twitter, LinkedIn o Facebook como otros proveedores más enfocados a un público en concreto como podría ser la suite de productos en “La Nube” de Zoho (CRM, Recruit) u otros como servicios de RSS proporcionan con mayor frecuencia APIS que los desarrolladores pueden utilizar para interoperar unos con otros. Esto también da lugar al desarrollo de widgets o componentes reutilizables que pueden ser llevados con mayor o menor facilidad a unas u otras páginas. Por ejemplo, si observamos la librería de módulos existentes para el CMS Drupal, podemos encontrar una enorme variedad de ellos cuya finalidad es implementar este patrón mediante la inclusión de widgets tales como el botón “Me Gusta” de facebook o un RSS de Twitter.

El patrón Experiencia Mejorada de Usuarios tiene como motivación tratar de buscar la mejora de la interacción entre el usuario y la web de forma que se mejore la experiencia del primero en términos de presentación y priorización de la información así como actualización, facilidad de uso e intuitividad.

En su propia definición se concluye que es un patrón tremendamente abstracto, que abarca la mejora del usuario en todos los aspectos posibles, desde muchos puntos de vista. Este patrón podría estar detrás de la adecuada elaboración de un formulario de tal forma que se auto

completen campos, se prediga lo que el usuario escriba y los campos no aparezcan a la vez sino que existe un flujo de navegación que no haga tedioso el proceso. En este caso el patrón abarcaría la utilización de varios patrones concretos.

Pero este patrón también podría abarcar el proceso de diseño de una web por completo, incluyendo las decenas de patrones que puedan ser utilizados, así como los flujos de información de la web ya que, por ejemplo, no es la misma experiencia de usuario tener que pasar por 3 páginas para editar sus datos de contacto que hacerlo sin recargar ni si quiera una página entera mediante el uso de AJAX.

El último de los patrones relacionados con RIA en el libro es el patrón Web sincronizada. En este patrón, cuando se habla de sincronización no se está refiriendo a la utilización o no de protocolos de comunicación síncronos entre la parte cliente de una aplicación y la parte servidor, como podría parecer.

El patrón de la web sincronizada se refiere a la sincronización del estado de la aplicación de tal forma que las aplicaciones web puedan ser utilizadas tanto de manera online cuando se dispone de conexión a internet como de manera offline si no se dispone de esta o no se desea utilizarla. De esta forma, el hecho de no tener conectividad con internet no es un absoluto impedimento para la ejecución de la aplicación, si bien es entendible que el grado de abstracción de este patrón permite imaginar muchas situaciones distintas.

La idea es que después de trabajar de manera offline el estado pueda sincronizarse cuando haya conexión.

Un ejemplo de implementación de este patrón es el reciente Google Drive. Esto es un servicio en “La Nube” de almacenamiento de documentos online, con el que se pueda acceder a cualquier documento desde cualquier lugar en cualquier momento o sincronizarlo en un disco duro local, por ejemplo.

También, Drive tiene la posibilidad de utilizar un plugin para el navegador de Google Chrome, con el que pueden editarse offline los documentos en el navegador y, posteriormente, sincronizarlos, sin tener que almacenar una estructura de carpetas y archivos en el disco local.

3.2.4 Catálogo de Scott y Neil

En su libro *Designing Web Interfaces* [3], Scott y Neil proponen un vasto catálogo de patrones de RIA para proporcionar buenas experiencias de uso a sus usuarios basándose en su amplia experiencia.

El catálogo de patrones que han elaborado se divide en seis principios que facilitan la organización:

Hacerlo directo, que propone formas de conseguir que los usuarios realicen acciones minimizando el número de pasos para ello.

Mantenerlo ligero, en el que se proponen varios mecanismos de herramientas contextuales y su utilización.

Quedarse en la página: propone minimizar el número de pantallas por las que tengan que pasar los usuarios para realizar acciones.

Proporcionar invitaciones: patrones para que el usuario pueda descubrir qué acciones puede llevar a cabo y cómo hacerlo.

Usar transiciones: donde explica el uso y beneficios de las transiciones visuales.

Reaccionar inmediatamente: patrones cuya finalidad es que la información de la web esté siempre actualizada sin necesidad de la interacción del usuario.

Este catálogo [3] es con mucho el más completo, y es el que se ha utilizado como referencia en este trabajo. En la Sección 5, se procede a una revisión detallada del mismo donde se hace un análisis del soporte a las tecnologías RIA a los patrones de dicho catálogo.

Capítulo 4 Extensión de UML WAE para el modelado de estas tecnologías

En un principio, UML estándar no puede ser utilizado directamente para caracterizar la capa de presentación de aplicaciones web. Sin embargo, la extensión de UML para aplicaciones web UML WAE, sí que puede ser extendido para su uso con aplicaciones RIA.

Esta sección define una extensión de UML WAE para caracterizar el diseño de aplicaciones RIA mediante el uso de UML. Al igual que sucede con UML WAE, el uso de esta extensión en el contexto de una arquitectura multicapa [48] sólo afectaría a la capa de presentación de la aplicación [49].

4.1 Javascript y HTML

En este trabajo se ha elegido el framework Javascript ExtJS como tecnología de referencia para explicar y ejemplificar el desarrollo de RIA con Javascript y HTML. Este framework pone a disposición del diseñador unas bibliotecas de funciones entre las que se encuentran funciones de utilidades o de manejadores de eventos así como extensas bibliotecas de objetos, tanto abstractos como concretos, para su uso directo en la elaboración del diseño de la aplicación.

Normalmente, una página HTML utilizará algunas de estas funciones en los manejadores de eventos de sus elementos como, por ejemplo, la invocación desde, por ejemplo, el manejador del evento `onClick` de un botón, de un método que encapsule una llamada asíncrona cuyo resultado sea la actualización de un campo de texto.

En otros muchos casos, una página cliente estará compuesta parcialmente por objetos Javascript que la doten, por ejemplo, de componentes visuales avanzados como tablas dinámicas o galerías fotográficas con transiciones. Estos componentes, en el caso de ExtJS, están organizados en bibliotecas (archivos *.js) que deberán ser importadas por la página. En el momento del diseño puede decidirse delegar todo el desarrollo de objetos así como su instanciación a un archivo externo a la página HTML o se puede hacer en la propia página. Para instanciar los objetos Javascript suele utilizarse el evento `onLoad` de la página. En él puede invocarse a las funciones ExtJS necesarias para la creación de objetos, manipular el DOM de dicha página y añadirlos.

Web Application Extension [7] define un perfil UML suficiente para modelar RIA que se vaya a implementar con tecnología Javascript y HTML. Los componentes del framework de referencia en este trabajo que se utilizan en el diseño de una aplicación web es están incluidos en el perfil UML: funciones, archivos, bibliotecas y objetos Javascript cliente.

En el ejemplo visto en el capítulo 2, se mostró el código de la figura 2.1 en la que encontramos los objetos javascript *BookStore*, *Book*, *BookGrid*, *BookDetail*, *BookMasterDetail*.

- BookMasterDetail es el contenedor externo de la tabla. Contiene un atributo de tipo BookGrid y otro de tipo BookDetail.
- BookGrid es la tabla en sí. Contiene una lista de Book y muestra sus atributos Author, Title, Manufacturer y Product Group en columnas.
- BookDetail es un panel que contiene la información del libro en modo texto y un enlace, que es otro atributo de Book.
- Book es la clase que se representa en esta tabla.

Además de la tabla debe existir una página HTML donde se incluyan estos objetos. Dados todos los elementos anteriores y utilizando WAE, el diseño sería el descrito en la figura 4.1.

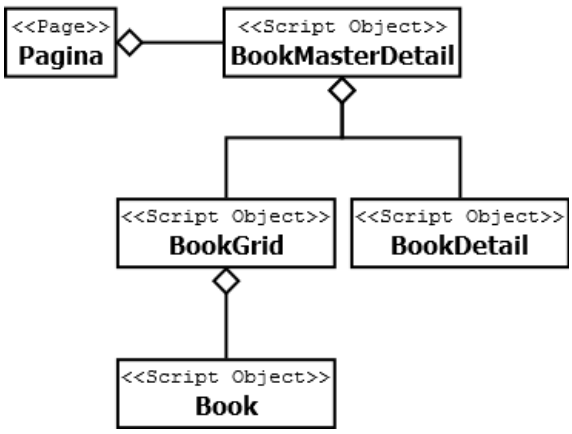


Figura 4.1 Diagrama de código ExtJS

4.2 Adobe Flex

Como se ha explicado anteriormente, las aplicaciones Flex se compilan en un archivo Flash que suele ser incrustado en una zona determinada de un documento HTML. En UML

WAE esta característica se modela como cualquier otro objeto que se utilice del mismo modo como, por ejemplo, un applet de java. Pero de la misma manera que desde el lenguaje ActionScript puede acceder al DOM de la página donde reside y manipularlo o ejecutar funciones Javascript incluidas, también se puede permitir el acceso a clases y métodos de ActionScript desde Javascript. Esto permite la creación de objetos Flex dinámicamente tanto dentro como fuera del ámbito de la propia aplicación Flex. Es labor del diseñador utilizar estas características con cuidado e incluso definir restricciones explícitas si es necesario para evitar situaciones complejas y de difícil mantenibilidad.

WAE no define explícitamente estereotipos para las clases que se utilizan en Flex así que en el segundo caso expuesto anteriormente no habría forma de definir una clase *AdvancedDataGrid* como la del ejemplo que vamos a utilizar para esta tecnología en la figura 4.2.

AdvancedDataGrid Control

Region	Territory	Territory Rep	Revenues	
			Actual	Estimate
Southwest	Arizona	Barbara Jennings	38865	40000
Southwest	Arizona	Dana Binn	29885	30000
Southwest	Central California	Joe Smith	29134	30000
Southwest	Nevada	Bethany Pittman	52888	45000
Southwest	Northern California	Lauren Ipsum	38805	40000

Figura 4.2 Ejemplo de AdvacedDataGrid, de Flex

Si esta tabla residiera en un único archivo compilado que se agregara a la página de forma estática y no se fuera a manipular desde código externo, podrían diferenciarse dos modelos de clases:

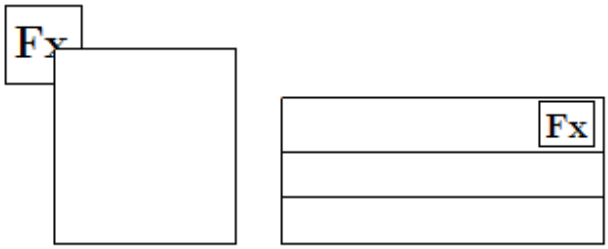
- El primero modelaría la página HTML y una agregación de un componente mediante la etiqueta <OBJECT>
- El segundo sería un diagrama UML con las clases que componen ese objeto Flash.

Dado que el desarrollo de la aplicación Flex y el código de la página no suelen ser en paralelo, esta aproximación no sería incorrecta puesto que podrían incluso ser equipos distintos los que realizaran ambos desarrollos.

No obstante, puede darse la situación de que el código Javascript de una página y/o el ActionScript de Flex manipulen objetos Flex o componentes de la página respectivamente o que el diseño deba reflejar simultáneamente características de ambas tecnologías. Para ese caso debería extenderse WAE con un nuevo estereotipo: Clase Flex Cliente.

La metaclase de Clase Flex Cliente sería Class. Una Client Flex Class es una clase estereotipada como <<Flex Object>> y se mapea directamente con una clase Flex definida en el código bien con ActionScript o con MXML. Un caso especial de esta clase sería la clase Script, que podría, como ocurre con código Javascript, simplemente contener funciones.

Como iconos o clases estereotipadas podrían utilizarse las descritas en las Figuras 4.3 y 4.4



Figuras 4.3 y 4.4 Iconos de las nuevas clases Flex

El ejemplo de la tabla dinámica que ya se ha explicado anteriormente podría modelarse utilizando este nuevo estereotipo, tal y como describe la figura 4.5.

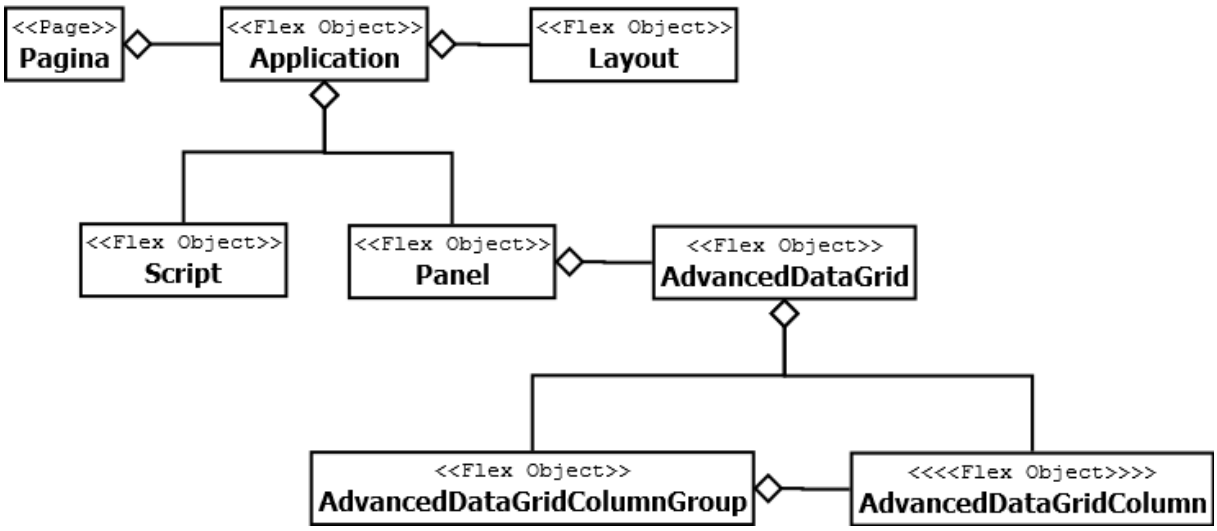


Figura 4.5 Diagrama del código Flex

4.3 Google Web Toolkit (GWT)

Una de las principales ventajas de diseñar una aplicación con GWT es la abstracción de los elementos relacionados con la web. Esta tecnología oculta los detalles que diferencian una aplicación de escritorio y una aplicación web: trata una página como una pantalla y los elementos estructurales (como los divs) son enmascarados como paneles, layouts, etc.

El desarrollo de la interfaz se hace directamente con código Java sin necesidad de utilizar HTML ni Javascript por lo que todo el código de la interfaz puede diseñarse de la misma manera que se diseña una interfaz de una aplicación de escritorio o cliente pesado utilizando, por ejemplo, swing.

Así como Swing ofrece una serie de clases que se utilizan para componer una interfaz, GWT proporciona sus propias clases java. La diferencia radica en la compilación: las clases swing se convertirán en bytecode al compilarse mientras que las clases de GWT se convertirán en Javascript y HTML.

El modelo WAE de una aplicación GWT es, en la mayoría de los casos, una única página HTML que referencia a varios archivos Javascript y CSS. Este diseño no aporta nada en absoluto al desarrollador, quien necesita un modelo de clases que le indique exactamente lo que debe desarrollar. Y este es un modelo UML que no necesita la extensión WAE puesto que no se utilizan estos elementos.

Pero, ¿qué ocurriría si se quisiera extender la funcionalidad de una de estas clases o implementar una nueva y para ello se requiriera utilizar código Javascript? GWT ofrece la posibilidad de utilizar código Javascript dentro de clases Java. El mecanismo para crear, por ejemplo, una nueva función Javascript que quedara encapsulada dentro de una función java, pasa por la utilización de *comentarios de código* que el compilador de GWT no omite, sino que lleva al código final sin traducir (recordemos que el compilador traduce Java a Javascript) estas sentencias. Este caso es muy concreto pero posible. No obstante esta situación no hace que se necesite utilizar WAE en todos los casos puesto que lo único que ocurre es que se están utilizando dos lenguajes de programación en vez de uno. La utilización de funciones Javascript que referencien a clases que se encuentran en archivos Javascript externos si podrían requerir el uso de los estereotipos definidos por WAE para librerías y objetos de script en caso de que el desarrollo no fuera trivial. Por ejemplo, si se define una función en una clase que utiliza instancias de una clase Javascript perteneciente a cierta biblioteca de objetos Javascript, habría que incorporar dicha dependencia en el diagrama de clases. Imaginemos un panel que vaya a contener una imagen a la cual quisiéramos incorporar la funcionalidad proporcionada por un objeto Javascript, como por ejemplo abrir esa imagen en una capa superpuesta con un fondo semitransparente que la realce (patrón O'Reilly "Dialog Overlay" con Lightbox Effect), tal y como describe la figura 4.6.

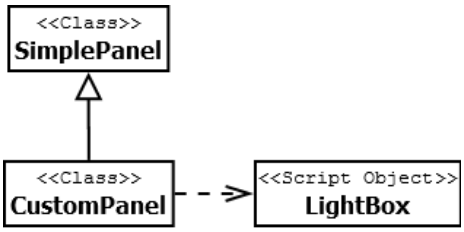


Figura 4.6 Diagrama de GWT

4.4 HTML 5

A la vista de las novedades estructurales de HTML 5, el perfil definido por WAE parece insuficiente. Si bien muchos de los cambios que vienen con la nueva versión no afectan tanto al

diseño como a la implementación, aquellos que podemos denominar como estructurales sí son importantes.

La posibilidad de definir secciones dentro de una página HTML así como establecer cabeceras y pies de página para cada una de ellas es una nueva característica que permite una mejor organización y estructura del documento. Sería importante tener en cuenta esta estructura a la hora de diseñar RIA y para ello se podría definir un nuevo estereotipo: HTML Section. Al haber a lo sumo sólo una cabecera y pie de página por sección podrían ser modelados como atributos de la sección no siendo necesario definir un estereotipo para ellos.

Los iconos para este estereotipo podrían ser los descritos en la figura 4.7.

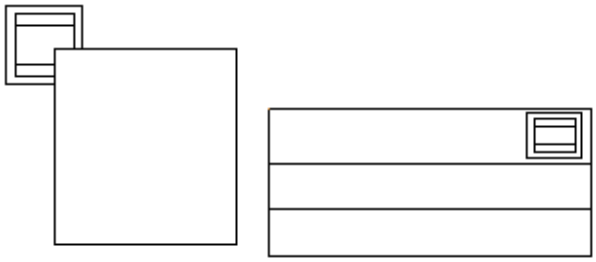


Figura 4.7 Iconos para estereotipo Section

Otros elementos que aparecen en la revisión del lenguaje de marcado y que presentan entidad suficiente como para necesitar un nuevo estereotipo son Audio, Video y Canvas. Audio y video son etiquetas muy similares que se utilizan para incorporar archivos multimedia a una página HTML sin recurrir a plugins como Flash. Con una simple etiqueta se puede añadir un archivo de audio o de video y con unas pocas líneas de Javascript se pueden utilizar botones para controlar la reproducción de los archivos multimedia. Los iconos para estos estereotipos, HTML Video podrían ser los descritos en la figura 4.8.

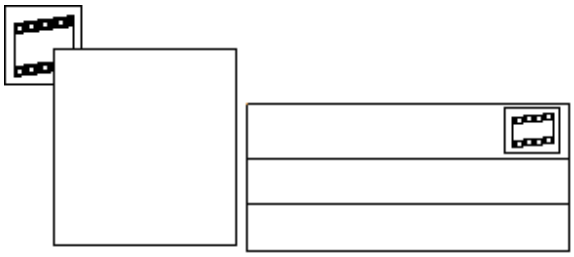


Figura 4.8 Iconos para estereotipo Video

Para modelar los elementos HTML Audio podrían utilizarse los iconos descritos en la figura 4.9:

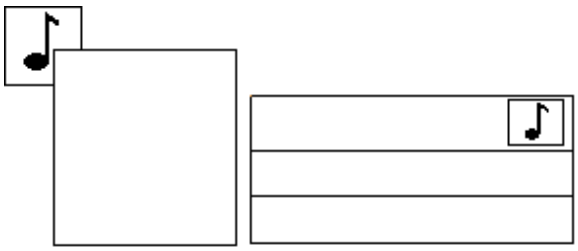


Figura 4.9 Iconos para estereotipo Audio

El elemento Canvas permite dibujar gráficos estáticos o animados mediante Javascript. La etiqueta Canvas simplemente define una zona de la página (mediante las propiedades Height y Width) donde se puede dibujar. El código Javascript (apoyado por las bibliotecas Javascript que van surgiendo) puede dibujar y animar el contenido en esa zona que, por supuesto, podría ser la totalidad del documento HTML. Utilizando capturas de eventos de teclado y de ratón puede utilizarse para reaccionar a las acciones de los usuarios, por ejemplo, para implementar juegos. Podría ser útil también definir un nuevo estereotipo para este elemento: HTML Canvas. Este estereotipo podría utilizar los iconos descritos en la figura 4.10.

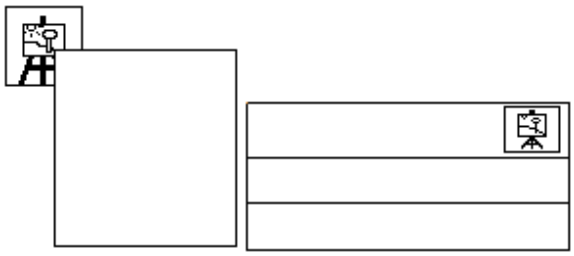


Figura 4.10 Iconos para estereotipo Canvas

A continuación, la figura 4.11 describe un ejemplo de diseño de página HTML con un elemento de video incrustado y una sección que contuviera a su vez un elemento Canvas.

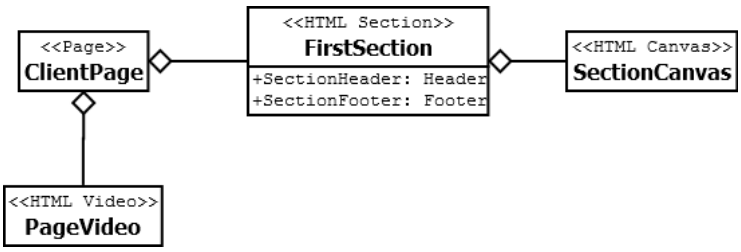


Figura 4.11 Ejemplo de página HTML

4.5 JSF

El mecanismo que utiliza JSF y que lo distingue del resto de tecnologías, es la existencia y el fácil uso para el desarrollador de los manejos de eventos y propiedades de lado servidor. Los Managed Beans representan atributos de elementos de las páginas XHTML o JSP de una aplicación JSF y contienen métodos que constituyen los manejadores de eventos de lado servidor. Los Managed Beans no están relacionados con una única página ni viceversa: una página puede utilizar atributos y manejadores de distintos beans así como distintas páginas podrían compartir el mismo Managed Bean,

Por esta razón el diseño de páginas JSF conlleva el modelado de páginas y Managed Beans por separado puesto que para el mismo requisito puede haber distintos diseños igualmente funcionales. Podría, por ejemplo, relacionarse cada página a un bean, o utilizar un bean distinto por cada clase de elemento que pueda haber en las páginas (un Managed Bean para botones, otro para listas desplegables, otro sólo para eventos…) y sería igualmente funcional si bien no muy ortodoxo.

Para modelar aplicaciones JSF se propone utilizar jerarquías de clases para los Managed Beans que, diseñadas correctamente, terminen en una colección de clases concretas que representen cada una a una página. De esta forma podría extenderse el perfil WAE con una única relación nueva, estereotipada como <<Managed>> que relacionaría las páginas con sus Managed Beans y con una nueva clase estereotipada, <<Managed Bean>>, cuyo icono propuesto representaría una marioneta que hace el papel de una página web, tal y como describe la figura 4.12.

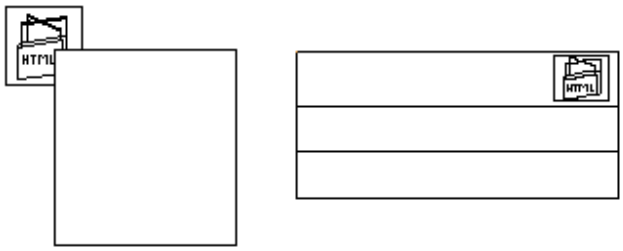


Figura 4.12 Iconos para estereotipo Managed Bean

Si dos páginas comparten atributos o manejadores de eventos deberían estar representados en superclases comunes a sus Managed Bean. Por ejemplo, si todas las páginas de una aplicación comparten una cabecera con un menú, debería existir un Managed Bean, probablemente abstracto, que contuviera los elementos básicos que compusieran ese menú y del que, a través de la jerarquía de clases, todas heredaran, tal y como describe la figura 4.13.

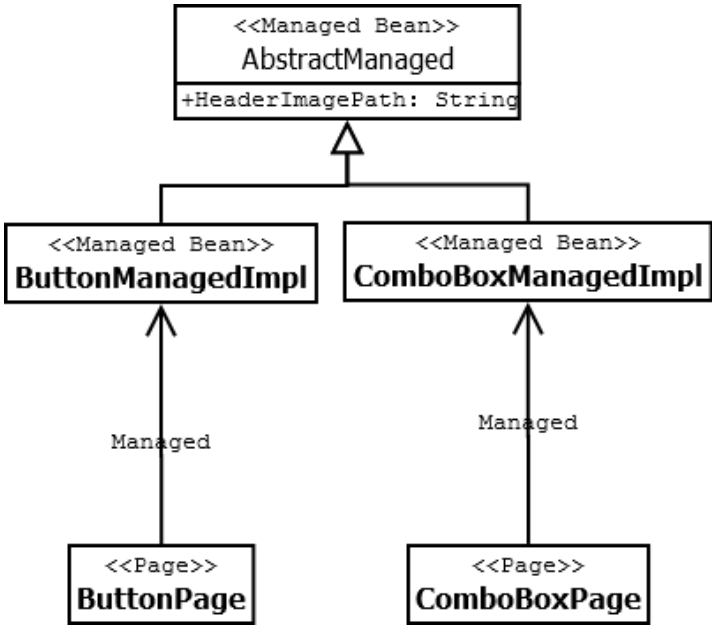


Figura 4.13 Diagrama de clases para JSF

Un aspecto importante de JSF es su gestión del flujo de navegación. El modelado del flujo de navegación en WAE queda oculto tras la lógica del controlador, de tal forma que no se puede expresar un direccionamiento condicional entre páginas dependiendo de, por ejemplo, el

valor de uno de los parámetros de la petición de request. Esta situación se puede definir fuera del controlador en JSF ya que en sus archivos de configuración existe la posibilidad de establecer mediante reglas condicionales el flujo de la presentación.

Por ello, también sería deseable ampliar WAE en este sentido. En dicho perfil UML encontramos el estereotipo <<forward>> que puede aplicarse a la delegación del proceso de una petición del cliente entre dos páginas de servidor o una página de servidor y una cliente. Si a este estereotipo se le añade una expresión condicional, podría diseñarse este flujo de navegación que ofrece JSF, tal y como describe la figura 4.14.

Sin embargo, esta aproximación dificultaría sobremanera la legibilidad de los modelos. Por tanto se propone una extensión de UML WAE similar a la descrita en trabajos como [49].

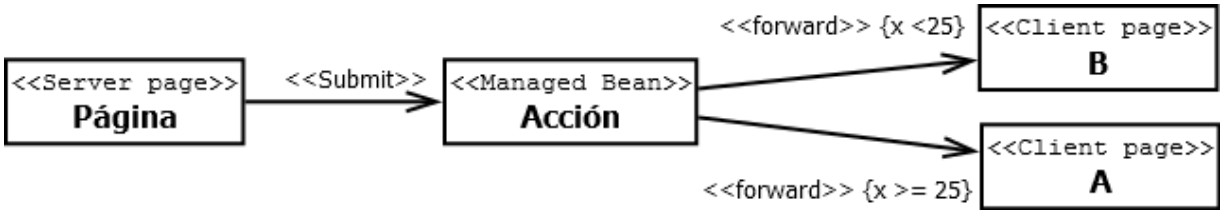


Figura 4.14 Extensión del estereotipo Forward

4.6 Conclusiones

Es factible el uso de el perfil UML WAE para las tecnologías basadas en Javascript y HTML y GWT sin necesidad de realizar ningún tipo de extensión ya que todos los elementos y relaciones ente clases y objetos ya están contemplados en él.

No obstante, el perfil carece de soporte para los objetos Flex por lo que debería incorporarse la extensión propuesta en el caso de querer modelar aplicaciones RIA cuya tecnología subyacente sea ésta.

Para el modelado de JSF se requiere extender el perfil con los tipos de clase Managed Bean y el tipo de relación que vincula una página con ella. Además, las relaciones forward podrían incorporar expresiones condicionales para poder modelar el flujo de navegación.

Finalmente, HTML 5 propone una serie de elementos novedosos y que deberían ser incorporados al perfil para que pudieran ser modelados. Estos elementos son Section, Video, Audio y Canvas.

Capítulo 5 Soporte de las tecnologías a patrones Scott y Neil

A continuación se presenta el estudio de la capacidad de las distintas tecnologías explicadas en los puntos anteriores de dar soporte a los patrones y principios de RIA descritos Bill Scott y Theresa Neil en su libro Designing Web Interfaces [3], así como los componentes de cada tecnología responsables de soportar a dichos patrones.

En el libro se presentan colecciones de patrones de diseño de interacción, organizadas a su vez en los seis principios mencionados en la Sección 3: hacerlo directo, mantenerlo ligero, quedarse en la página, proporcionar invitaciones, usar transiciones, y reaccionar inmediatamente.

Las colecciones de patrones los agrupan según criterios de semejanza dentro de los principios básicos en los que operan y, precisamente por esa razón, en muchos casos se van a analizar en conjunto.

Los patrones: *Edición en la página, arrastrar y soltar* y *selección directa* pertenecen al principio 1: *hacerlo directo* mientras que el patrón *Herramientas contextuales* pertenece al principio 2: *Mantenerlo ligero.*

Los patrones *capas superpuestas, capas internas, páginas virtuales* y *flujo del proceso* pertenecen al principio 3: *quedarse en la página*

El principio 4, *proporcionar invitaciones*, está formado por los patrones *invitaciones dinámicas* e *invitaciones estáticas.* El principio 5, *usar transiciones*, engloba al patrón *Patrones transicionales*,

Finalmente, el principio 6, *Reaccionar inmediatamente*, está compuesto por los patrones *de búsqueda*, y *de retroalimentación*

Salvo casos excepcionales, los patrones dentro de una misma colección tienen diferencias en el enfoque, funcionalidad o intención más que en el tipo de componente utilizado, por lo que el soporte a los patrones se puede dar de igual manera y suele ser sólo la configuración y contexto del componente el que marque la diferencia.

El soporte a un conjunto de patrones puede ser total, parcial, nulo o no aplicable.

- Se considera que la tecnología soporta totalmente una colección de patrones si da soporte a todos elementos que los que la componen. Por ejemplo, ExtJS es capaz de soportar totalmente todos los patrones de “Capas superpuestas” ya que tiene un elemento, window, adaptable a las situaciones descritas.

- La tecnología soportará parcialmente el conjunto de patrones si algunos de ellos pueden implementarse y otros no. Por ejemplo, el patrón "interfaz de usuario ampliable/reducible" es soportado parcialmente por GWT ya que no todos sus componentes son capaces de soportarlo.
- Cuando la tecnología no soporte ninguno de los patrones se considerará nulo.
- Si la tecnología no está pensada para dar soporte a un tipo de patrón concreto se considerará no aplicable. Esto aplica, por ejemplo, a HTML 5 cuando los patrones se refieran a la funcionalidad de *widgets* o componentes desarrollados en un framework, ya que HTML 5 no es un framework.

Para cada patrón o colección de patrones que se estudian se presenta una tabla donde se relaciona la tecnología, el grado de soporte que da, el componente o componentes que lo dan y unas observaciones en caso de que procedan.

5.1 Edición en la página

Esta colección de patrones agrupa las distintas técnicas de diseño web que cambian el enfoque clásico de los sitios web tradicionales en los que para editar el contenido de algún elemento de la página implica navegar a una página de diseño por un enfoque en el que el propio componente que se desea editar cambia su estado y permite su edición en la misma página en la que se muestra. En la Tabla 5.1 se puede ver el grado de soporte de las tecnologías a este patrón.

Tabla 5.1 Soporte de las tecnologías al patrón Edición en la página

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Forms, Window, Dialog, Editable Grid	
Flex	Total	RichEditableText, DropDownList, Panel, DataGrid	
GWT	Total	BasicText, ListBox, DialogBox, DataGrid,	
HTML 5	Nulo		Los componentes de HTML 5 necesitan un

			desarrollo en Javascript considerable para poder dar soporte a los patrones
JSF	Total	inplaceInput, inplaceSelect, popupPanel, panelGrid	

5.2 Arrastrar y soltar ó "*Drag and drop*"

En la tabla 5.2 se indica el grado de soporte de estos patrones, que exploran el uso del ratón para arrastrar elementos de una web como se hace con los archivos y carpetas de un sistema operativo o con elementos de aplicaciones de escritorio. Esta forma de interactuar con la web proporciona funcionalidades como personalizar el aspecto de la web, ordenación de listas de forma muy intuitiva, creación de colecciones de objetos (como un carrito de la compra) o la ejecución de acciones determinadas como enviar un elemento a una papelera (eliminarlo).

Tabla 5.2 Soporte de las tecnologías al patrón Arrastrar y soltar

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Eventos onNodeEnter, onNodeOut, onNodeOver, onNodeDrop	
Flex	Total	Propiedades dropEnabled, dragEnabled, dragMoveEnabled	
GWT	Total	PickupDragController, IndexedDropController y método	

		makeDraggable.	
HTML 5	Total	Atributo draggable y eventos dragover, dragenter, dragleave, drop…	
JSF	Total	dragSource, dragTarget, accepType, dragIndicator…	

5.3 Selección directa

Los tres patrones que recoge Selección directa están enfocados a la forma en la que se pueden seleccionar elementos de una web. En RIA, podemos seleccionar objetos como imágenes u otro tipos de elementos haciendo clic directamente sobre ellos o utilizando otros elementos como *CheckBoxes*. También se pueden ir añadiendo a una lista auxiliar a medida que se van seleccionando de una forma u otra.

Tabla 5.3 Soporte de las tecnologías al patrón Selección directa.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	CheckBox, DataView.	
Flex	Total	List, DataGrid, CheckBox	
GWT	Total	CheckBox, DataGrid	
HTML 5	Total	CheckBox, Table	Requiere Javascript
JSF	Total	ExtendedDataTable, CheckBox	

5.4 Herramientas contextuales

Estos patrones definen distintas formas de mostrar herramientas a los usuarios de una web sin especificar concretamente estas herramientas ya que pueden ser de multiples tipos

básicos que van a estar soportados por todas las tecnologías. Los diferentes enfoques, en cambio, sí especifican elementos que influyen en la forma de mostrar estas herramientas: eventos hover, capas superpuestas, eventos clic de botón derecho.

Tabla 5.4 Soporte de las tecnologías al patrón Herramientas Contextuales.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	ContextMenu, Panel	
Flex	Total	MenuItem, evento Show()	
GWT	Total	PopupPanel, ContextMenuEvent	
HTML 5	Total	Context Menu, Div, onMouseOver.	
JSF	Total	ContextMenu, onmouseover	

5.5 Capas superpuestas

Los patrones de interacción clasificados como Capas superpuestas utilizan los eventos *clic* y *onmouseover* junto a elementos que muestran capas superpuestas sobre la página actual para conseguir destacar algo en concreto. Esto puede utilizarse, por ejemplo, para ver los detalles de un libro o superponer un formulario de contacto al pulsar sobre un enlace determinado en lugar de cambiar de página. También se puede utilizar para realizar un proceso más complejo en el que se involucren varias capas superpuestas, como el de registro en la web. Adicionalmente se pueden utilizar capas con grado de transparencia para oscurecer toda la zona que no se quiera destacar.

Tabla 5.5 Soporte de las tecnologías al patrón Capas superpuestas.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Window	
Flex	Total	Panel	
GWT	Total	PopupPanel,	

HTML 5	Total	Div	
JSF	Total	PopupPanel	

5.6 Capas internas

Estos patrones se corresponden a los del apartado anterior pero sin utilizar capas superpuestas. Esto puede ser beneficioso en muchos casos porque las capas ocultan información que puede necesitar el usuario o simplemente porque interrumpe el flujo de acciones que esté tomando. Para dar soporte a estos patrones suelen ser necesarios paneles que se oculten y muestren bajo demanda del usuario, paneles de tipo acordeón y paneles de pestañas.

Tabla 5.6 Soporte de las tecnologías al patrón Capas internas.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	TabPanel, Accordion, Panel	
Flex	Total	Accordion, TabNavigator, Panel	
GWT	Total	TabLayoutPanel, StackLayoutPanel, Panel.	
HTML 5	Total	Div, Details, Article, Section	
JSF	Total	TabPanel, Panel, Accordion	

5.7 Páginas virtuales

Las técnicas que sugieren estos patrones tienen como finalidad aumentar la cantidad de contenido que se puede visualizar en una página mediante distintas técnicas. Estas técnicas son bastante distintas entre sí y se van a tratar por separado.

5.7.1 Desplazamiento virtual y Carrusel

Estos patrones utilizan un área de la página dispuesta en forma de lista y que cuenta con una barra de desplazamiento vertical. En esta área se pueden precargar todos los elementos que se quieran mostrar o ir cargándolo bajo demanda a medida que el usuario va desplazándose por la lista

Tabla 5.7 Soporte de las tecnologías a los patrones Desplazamiento virtual y Carrusel.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Grid, PagingScroller.	
Flex	Total	DataGrid, DataGroup.	
GWT	Total	DataGrid	
HTML 5	Nulo		Requiere de librerías Javascript externas para dar soporte.
JSF	Total	ExtendedDataTable, DataGrid.	

5.7.2 Paginación interna

Propone un mecanismo en el que la paginación de una gran colección de elementos se realice en una zona de la página. Es semejante a Desplazamiento virtual y Carrusel, pero se diferencia en que utiliza un paginador para realizar el cambio de una página a otra y las páginas tienen un número determinado de elementos.

Tabla 5.8 Soporte de las tecnologías al patrón Paginación interna.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Grid, Paging.	
Flex	Total	ScrollBar, Form, Button…	No hay un widget implementado, pero se puede desarrollar.
GWT	Total	DataGrid	

HTML 5	Nulo		Requiere de librerías Javascript externas para dar soporte.
JSF	Total	DataGrid, DataScroller.	

5.7.3 Interfaz de usuario ampliable/reducible

Este patron es el menos común de todos los que conforman la categoría. La filosofía de paginación virtual de este patrón es utilizar una función de ampliación y navegación sobre una página de grandes dimensiones en la que pueda incorporarse mucho contenido.

Tabla 5.9 Soporte de las tecnologías al patrón Interfaz de usuario ampliable/reducible.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Parcial	Draw.Component	Soporta Zoom para imágenes.
Flex	Total	MouseWheelZoomControl, LayoutAnimator, LayoutTarget	
GWT	Parcial	Animation	Sólo zoom en imágenes.
HTML 5	Parcial	Canvas	El elemento canvas soporta la función setScale, que permite hacer zoom sobre él.
Rich Faces	Nulo		Incluso el zoom para imágenes requiere Javascript externo.

5.8 Flujo del proceso

Recoge un conjunto de técnicas que explotan los mecanismos de RIA para mejorar la experiencia del usuario de una página cuando, por ejemplo, realiza un proceso de compra. Estos patrones se centran en la idea de que el flujo de información debe mantenerse en una única página de tal forma que acciones que puedan servir como criterios de filtrado para alcanzar un resultado final se vean llevadas a cabo y sus resultados reflejados sin cambiar de página.

Por ejemplo, si un cliente desea comprar un pantalón en una tienda online, puede que al elegir el color esté eliminando las tallas 48, 50 y 52 por criterios de stock. Esto podría reflejarse en la misma página deshabilitando los controles que procedan, por ejemplo, unos RadioButton que permitan elegir la talla, lo cual es bastante sencillo de implementar con todas las tecnologías que se estudian en el presente trabajo.

Para dar soporte a estos patrones sólo es necesaria la capacidad de realizar llamadas asíncronas y modificar el DOM de la página (o el estado de los objetos en el caso de Flex) una vez recibida la respuesta, capacidad que todas las tecnologías propuestas poseen.

Debe tenerse en cuanta de que, aunque todas las tecnologías den soporte a todos los patrones, podría darse el caso de querer implementar un tipo de flujo de proceso que una de ellas no pudiera. Esto es porque, en este caso, los patrones son muy poco específicos con respecto a cómo debe realizarse la implementación: son más abstractos y dan mayor flexibilidad para el diseñador. No obstante, podría definirse otra forma en la que implementar el patrón y que la tecnología sí pudiera llevarla a cabo. Por ejemplo, imaginemos que en un determinado flujo definido para una página se desea que el usuario vea sus datos y, en caso de haber un error en ellos, los edite directamente sin tener que cambiar de página. En este caso no podríamos hacerlo con HTML 5 sin recurrir a realizar un cierto desarrollo con Javascript.

5.9 Invitaciones estáticas

Como en el caso anterior, estos patrones son muy abstractos y no especifican como deben llevarse a cabo por lo que no se puede discriminar qué tecnologías pueden llevarlos a cabo o no hasta que no se estudie el caso concreto que se quiera abordar. De forma abstracta, todas las tecnologías pueden dar soporte a estos patrones que simplemente requieren, como mucho, la carga asíncrona de contenido en la página y esta es una característica básica de todas ellas.

Existen dos patrones de invitaciones estáticas. La primera es “Llamada a la acción” y consiste en comunicar le al usuario que se encuentra en la página web qué acciones puede llevar a cabo y cómo puede ser capaz de hacerlo de manera explícita mediante mensajes estáticos, imágenes, botones, etc.

En segundo lugar se encuentra “Invitar a un tour” que consiste en integrar una ayuda guiada a los usuarios con el sitio web donde se muestren ejemplos y los usuarios vayan realizando ciertos pasos, a modo de tutorial para la propia página. Se recomienda su uso en páginas que redefinen su interfaz, por ejemplo.

5.10 Invitaciones dinámicas

Las invitaciones dinámicas consisten en una colección de patrones que persiguen el objetivo de ayudar al usuario sin estar constantemente en la propia página sino aparecer en momentos puntuales.

Las invitaciones dinámicas pueden ser desde etiquetas de texto flotante hasta menús que aparezcan al mover el ratón sobre un elemento de la página, pasando por indicaciones que puedan aparecer en la pantalla a medida que el usuario esté intentando realizar una acción de Drag and drop (por ejemplo, indicarle que un contenedor es válido para soltar el elemento y otro no)

5.10.1 Invitación “Más contenido”

El patrón consiste en el uso de Páginas virtuales para que el usuario pueda navegar por más productos, categorías u otro tipo de elementos, por lo que el soporte a este patrón coincide con lo establecido en el punto 5.7 del presente trabajo.

5.10.2 Invitación a “Drag and drop”

Este patrón requiere la posibilidad de mostrarle al usuario ciertas indicaciones antes y durante el proceso de “Arrastrar y soltar” de algún elemento de la página: indicar elementos susceptibles de ser arrastrados e indicar si el proceso se está realizando adecuadamente mediante etiquetas flotantes. El soporte a este tipo de patrón es coincidente con la tabla del punto 5.2.

5.10.3 Invitaciones al pasar sobre un elemento, a interactuar con elementos y de inferencia

Estos patrones se implementan utilizando los eventos onmouseover y distintos widgets más o menos complejos dependiendo de las necesidades que se tengan que cubrir.

Tabla 5.10 Soporte de las tecnologías a los patrones Invitaciones al pasar sobre un elemento, a interactuar con elementos y de inferencia.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	ToolTip	
Flex	Parcial	Tooltip	No permite incorporar otros elementos dentro.
GWT	Total	PopupPanel	
HTML 5	Total	Los necesarios: Div, A, Button ya que el patrón es muy genérico y deja a la imaginación de los diseñadores de usabilidad los elementos que se utilicen para implementarlo.	Apoyado en el uso de CSS (evento hover, propiedad visibility)
Rich Faces	Total	ToolTip	

5.11 Patrones transicionales

Esta colección de patrones engloba distintos efectos especiales que pueden utilizarse en la web con distintas finalidades.

5.11.1 Iluminar y oscurecer

Este patrón consiste en el uso de capas con cierto nivel de transparencia para oscurecer parte o la totalidad de la página en favor de una zona que queda sin ocultar parcialmente o bien una capa superpuesta a la oscurecida.

Tabla 5.11 Soporte de las tecnologías al patrón Iluminar y oscurecer.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Spotlight	
Flex	Total	PopupPanel	
GWT	Total	DialogBox	
HTML 5	Nulo	PopupPanel	Utilizando CSS3 se puede conseguir.
Rich Faces	Total	<rich:jQuery>	Permite invocar funciones Javascript.

5.11.2 Expandir y contraer

Se puede utilizar conjuntamente a los patrones de la sección 5.6 para que los paneles que se abren o cierran bajo demanda del usuario lo hagan con una animación en la que se expanden o contraen.

Tabla 5.12 Soporte de las tecnologías al patrón Expandir y contraer.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Panel	
Flex	Total	Panel	
GWT	Total	TabPanel, PopupPanel, DisclosurePanel…	
HTML 5	Nulo		Utilizando CSS3 se puede conseguir.
Rich Faces	Total	Effect	

5.11.3 Desvanecimiento

Cuando se eliminan elementos de una lista o colección puede utilizarse este patrón que sugiere un desvanecimiento progresivo del elemento previo a su desaparición completa de la página.

Tabla 5.13 Soporte de las tecnologías al patrón Desvanecimiento.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Propiedades FadeIn y FadeOut	
Flex	Total	Fade	
GWT	Total	Accediendo a estilos y propiedades HTML y CSS	
HTML 5	Nulo		Utilizando CSS3 se puede conseguir.
Rich Faces	Total	Effect	

5.11.4 Animación

Este patrón se recomienda para clarificar las acciones que lleva a cabo un usuario en una página. La idea es que cuando se realiza una acción mediante Drag and drop, luego esa acción se repita rápidamente para que el usuario vea una reproducción de la acción que acaba de hacer.

Tabla 5.14 Soporte de las tecnologías al patrón Animación.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Ext.util.Animate	
Flex	Total	Animation	
GWT	Total	Accediendo a estilos y propiedades HTML y CSS	
HTML 5	Nulo		Utilizando CSS3 se puede conseguir.
Rich Faces	Total	<rich:jQuery>	

5.11.5 Punto de luz

El patrón Punto de luz consiste en destacar una zona de la pantalla durante unos segundos para atraer la atención del usuario, por ejemplo si hay alguna notificación que deba leer o si se ha añadido una nueva sección en la página.

Tabla 5.15 Soporte de las tecnologías al patrón Iluminar y oscurecer.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Ext.util.Animate	
Flex	Total	CrossFade	
GWT	Total	Accediendo a estilos y propiedades HTML y CSS	
HTML 5	Nulo		Utilizando CSS3 se puede conseguir.
Rich Faces	Total	Effect	

5.12 Patrones de búsqueda

Estos patrones son aplicables a los procesos de búsqueda que pueden encontrarse en sitios web como motores de búsqueda o tiendas online. Sugieren distintas aproximaciones que pueden aplicarse conjuntamente.

5.12.1 Autocompletar y sugerencias instantáneas

Estos patrones son muy semejantes y ambos se apoyan en la idea de ayudar al usuario a encontrar términos de búsqueda y disminuir la escritura. Para ello, se van sugiriendo terminaciones a los caracteres que va escribiendo el usuario en un cuadro de búsqueda (Auto complete) y se predicen las próximas palabras que va a escribir (Live suggest).

Ambos patrones pueden implementarse con cualquiera de las tecnologías propuestas en este trabajo ya que, aparte del motor de sugerencias, las características técnicas para implementarlo son sencillas y podrían realizarse con HTML y pocas líneas de Javascript.

Auto Complete suele venir incoporado en algunos widgets de frameworks, como en Rich Faces (AutoComplete), GWT (AutoComplete) y ExtJS (ComboBox).

5.12.2 Búsqueda instantánea

Conjuntamente con Auto complete y Live suggest puede utilizarse Live search. Este patrón es muy sencillo y propone que, a medida que el usuario va escribiendo términos en el cuadro de búsqueda, el panel donde aparecen estos resultados se va actualizando automáticamente, sin esperar al evento click del botón que envía la búsqueda o de la tecla enter, por ejemplo.

Como en el caso anterior, la parte visual de este patrón es muy sencilla y requiere tan solo manejar correctamente eventos de teclado para implementarlo.

5.12.3 Refinar la búsqueda

Este patrón también es muy común en tiendas online (artículos, subastas, viajes…) y consiste en la utilización de unos criterios para filtrar los resultados de una búsqueda una vez se ha realizado.

Tabla 5.16 Soporte de las tecnologías al patrón Refinar la búsqueda.

Tecnología	Grado de soporte	Componentes	Observaciones
ExtJS	Total	Grid Filter	
Flex	Total	AdvancedDataGrid	
GWT	Parcial	DataGrid	Permite ordenación pero no filtrado total.
HTML 5	Nulo		
Rich Faces	Total	DataTable	

5.13 Patrones de retroalimentación

Son cuatro patrones que se dividen en dos categorías atendiendo a su nivel de abstracción. Todo ellos están relacionados con la posibilidad de mostrarle al usuario el estado de la página en todo momento, tanto como respuesta a sus acciones como a las de otros usuarios.

5.13.1 Vista preliminar, descubrimiento progresivo e indicador de progreso.

Estos tres patrones tratan tres aspectos distintos de la actualización de la página según el usuario va llevando a cabo acciones en ella. Utilizando la carga asíncrona y manipulación del contenido se pueden modificar elementos como respuesta a la interacción del usuario.

El patrón Vista preliminar apuesta por mostrar el resultado de las acciones del usuario antes de que las realice, por ejemplo mostrando una lista de la cantidad de resultados que va a devolver su búsqueda atendiendo a los criterios que está utilizando.

Por otra parte, Descubrimiento progresivo sugiere que los elementos de la página no estén siempre visibles en todo momento, sino que vayan apareciendo a medida que el usuario necesite utilizarlos. Por ejemplo, si se está rellenando un formulario de registro en una web, podría ocultarse el campo “repetir contraseña” hasta que el usuario no haya rellenado el campo “contraseña”.

Finalmente, el patrón Indicador de progreso aconseja que se muestre algún tipo de indicación al usuario de que la acción que acaba de realizar está siendo interpretada y llevada a cabo, para que no crea que la página se ha quedado bloqueada o no responda en caso de que la acción lleve cierto tiempo de proceso.

5.13.2 Refresco periódico.

Como su propio nombre indica, este patrón consiste en que algunos elementos de la página actualicen su estado automáticamente cada cierto tiempo. Por ejemplo, si un post de un blog permite comentarios podría actualizarse cada minuto el panel donde se encuentran para que se vayan incorporando los que se vayan escribiendo.

Aunque es un poco más concreto, este patrón sólo requiere, al menos, la posibilidad de realizar peticiones asíncronas al servidor y manipulación de la página en la respuesta, algo que todas las tecnologías a estudio soportan.

5.14 Conclusiones

De los datos anteriores se puede extraer que todas estas tecnologías son muy interesantes debido a sus amplias capacidades, pero ninguna de ellas soporta la totalidad de los patrones descritos. Si bien no siempre se dará la situación en la que todos estos patrones sean utilizados en el mismo sitio web, hay que tener en cuenta que estas tecnologías no son suplementarias sino que, en muchos casos, se pueden combinar para hacer más rica la experiencia del usuario. Por ejemplo, es común utilizar objetos Flex en páginas que utilicen cualquier otra tecnología. Otra buena práctica sería utilizar siempre como base HTML 5 en lugar de HTML debido a todos los patrones soportados de forma nativa.

Hay tres tecnologías que ofrecen soporte, aunque sea parcialmente a todos los patrones: ExtJS, Flex y GWT. En cambio HTML 5 y JSF no tienen soporte a, al menos, un patrón.

De las tres tecnologías que ofrecen el mayor soporte, GWT presenta una deficiencia ligeramente mayor debido a no soportar totalmente el patrón “Refinar la búsqueda” que, en caso de aplicaciones en las que sea crítico implementarlo, su desarrollo podría mostrar mayor complejidad. No obstante esta tecnología se actualiza constantemente y se puede observar una mejora progresiva rápida que podría solucionar esta situación en un corto plazo de tiempo. La tecnología GWT es muy conveniente para muchos desarrollos, sobre todo en aquellas situaciones en las que los desarrolladores conozcan Java y desconozcan otras tecnologías.

Entre ExtJS y Flex hay dos diferencias en el soporte: Flex es la única de todas que es capaz de soportar el patrón “interfaz de usuario ampliable / reducible” y esto podría hacerla la más completa de no ser porque no soporta de forma nativa la incorporación de elementos básicos en los *Tooltips*, como el resto de tecnologías. No obstante existe una solución a esto que pasa por la utilización de paneles a los que asignan propiedades y eventos que simulen ser *Tooltips*. ExtJs sería más recomendable para usuarios con considerable experiencia en Javascript y frameworks asociados y Flex para proyectos ajustados de tiempo con desarrolladores hábiles o proyectos donde no importe que los usuarios tengan que descargar el plugin de flash. La curva de aprendizaje es similar en ambos.

Rich faces es una tecnología muy apropiada para todo tipo de proyectos, excluyendo únicamente aquellos que requirieran el uso de la interfaz de usuario ampliable y reducible. Hay que tener en cuenta que da soporte total al resto de patrones.

Finalmente HTML 5 queda muy lejos de ser tan potente como el resto de tecnologías pero es muy interesante como base del desarrollo. La venida de HTML 5 cambia las bases en las que sustentan estos frameworks, salvo Flex, haciendo que algunas de sus funciones sean más sencillas o eficientes por el soporte nativo que ya tiene a los patrones. Además, para proyectos en los que sólo se requieran los patrones que soporta, sería la primera opción a elegir debido a no tener que arrastrar dependencias y la facilidad de uso: es un lenguaje al que todos los programadores web están acostumbrados.

Capítulo 6 Conclusiones y trabajo futuro

A pesar de que las tecnologías RIA proporcionan una interacción con el usuario más cercana a la disponible en aplicaciones de escritorio, y por tanto, más potente y amigable, su uso en aplicaciones web no debe darse por descontado.

Por un lado la tecnología, en contraposición a otras tecnologías como HTML o XML, sigue siendo muy dependiente de proveedores concretos, vinculando la aplicación a marcos específicos.

Por otro lado, la profusión de tecnologías existentes, hace difícil encontrar programadores con experiencias en todas ellas. Precisamente, trabajos como el presentado en esta memoria, pueden ayudar a seleccionar tecnologías concretas RIA en función del proyecto considerado ya que se ha relacionado cada tecnología con el soporte específico a los patrones de diseño Scott y Neil.

Los patrones de RIA de Scott & Neil abarcan un abanico muy amplio de situaciones comunes en el desarrollo de este tipo de aplicaciones y el estudio del soporte que las tecnologías les dan es decisivo para la selección de una u otra. En general se ha comprobado que todas las tecnologías salvo HTML5 dan soporte a la gran mayoría de estos patrones por lo que, salvo en casos muy concretos, podríamos utilizar ExtJS, GWT, Flex o JSF para nuestros desarrollos. Es importante recalcar que estas tecnologías no son suplementarias sino que se pueden combinar para mejora la experiencia del usuario. En resumen, las más completas son ExtJS, GWT y Flex, aunque JSF también podría utilizarse en casi todos los escenarios posibles.

El caso de HTML5 es especial ya que, en realidad, está concebido para ser utilizado conjuntamente con CSS3 y código Javascript y en este trabajo no se ha estudiado en conjunto con CSS3 ni con librerías como JQuery puesto que ese caso ya está cubierto por ExtJS. Aun así, algunos patrones de RIA pueden implementarse directamente con HTML5 y en otros casos este lenguaje facilita en gran medida su implementación por lo que siempre sería recomendable utilizarlo.

Con respecto al soporte que las distintas notaciones de modelado proporcionan a las tecnologías RIAS cabe destacar dos tipos de notaciones. Unas, que han incorporado primitivas RIA a sus primitivas de modelado, las cuales se centran en modelar aplicaciones, en vez de código. Así, a pesar de soportar las principales construcciones RIA, proporcionan modelos que,

básicamente, sólo pueden ser convertidos en código a través de las herramientas CASE incluidas en las metodologías anexas a la notación.

Por otro lado, UML WAE la notación más cercana al uso tradicional de UML no incluye soporte a las tecnologías RIA. Este trabajo ha extendido UML WAE con el fin de que sea capaz de soportar los componentes de las principales tecnologías RIA.

Así, se han definido perfiles UML, integrables con UML WAE, para dar soporte a las principales tecnologías RIA.

Tanto en el caso de Javascript y HTML como en el de GWT se ha visto que no es necesario el uso de ningún perfil UML nuevo ya que UMLWAE da soporte a ellas.

En cambio, JSF requiere la modificación de la relación de relaciones y la inclusión de nuevos estereotipos. En HTML 5 y Flex también ha sido necesario la inclusión de nuevos estereotipos para describir las principales construcciones RIA.

Con respecto al trabajo futuro, en el momento de la elaboración de esta memoria, HTML5 aún no es estándar sino un borrador muy avanzado. Si bien algunas de las funcionalidades de este nuevo lenguaje ya están siendo soportadas por algunos navegadores aún puede haber lugar a cambios que afecten al contenido de este trabajo. Así, por ejemplo, hoy en día está abierta la discusión a incorporar mecanismos de transcripción para los videos y otra para proporcionar un API de acceso específica del elemento `Canvas` que sirvan para facilitar la lectura de texto que se genere en su interior ya que no es accesible para personas invidentes.

Con respecto a la notación de modelado, se ha optado, en un primer paso, por incluir perfiles específicos para tecnologías RIA. En un segundo paso, cabe abstraer estos perfiles específicos en un perfil genérico, que propicie el uso de un desarrollo dirigido por modelos, para generar diseños para tecnologías concretas.

Finalmente, las aplicaciones RIA están llegando poco a poco hasta los dispositivos móviles tales como teléfonos o tabletas. En el mercado existe bastante disparidad entre las distintas plataformas que, además, requieren un aprendizaje específico para el desarrollo de aplicaciones lo cual, como sucede con las distintas implementaciones de RIA, requiere un amplio conocimiento en cada tecnología (Android, iOS, BlackBerry, Windows Phone) y obliga a realizar varios desarrollos distintos para llevar la misma aplicación a todos los terminales.

Esta situación está haciendo que cada vez sea más frecuente el uso de RIA en este tipo de dispositivos que, afortunadamente, cuentan con navegadores de internet capaces de interpretar código HTML y Javascript.

El uso de RIA tiene su justificación debido a que se pueden desarrollar aplicaciones muy interactivas, casi tanto como las aplicaciones nativas desarrolladas con las herramientas propias para cada dispositivo, pero con la ventaja de que con un solo desarrollo se puede utilizar esa misma aplicación en todas las plataformas.

No obstante el uso de RIA no tendría tanto sentido si se hiciera para una sola plataforma ya que en ese caso sería recomendable realizar un desarrollo específico para la plataforma nativa del dispositivo.

Por ello, se espera un trabajo futuro centrado en el estudio de patrones RIA y/o marcos específicos para dispositivos móviles.

Referencias

[1] Nora Koch, Matthias Pigerl, Gefei Zhang, Tatiana Morozova: Patterns for the Model-Based Development of RIAs. ICWE 2009: 283-291

[2] James Governor, Dion Hinchcliffe, Duane Nickull: Web 2.0 Architectures - What entrepreneurs and information architects need to know. O'Reilly 2009: I-XXI, 1-247

[3] Bill Scott, Theresa Neil: Designing Web Interfaces - Principles and Patterns for Rich Interactions. O'Reilly 2009: I-XIX, 1-30945.

[4] Marino Linaje Trigueros, Juan Carlos Preciado, Fernando Sánchez-Figueroa: Engineering Rich Internet Application User Interfaces over Legacy Web Models. IEEE Internet Computing 11(6): 53-59 (2007)

[5] Alessandro Bozzon, Sara Comai, Piero Fraternali, Giovanni Toffetti Carughi: Conceptual modeling and code generation for rich internet applications. ICWE 2006: 353-360

[6] Santiago Meliá, José Javier Martínez Domene, Álvaro Pérez, Jaime Gómez: OOH4RIA Tool: Una Herramienta basada en el Desarrollo Dirigido por Modelos para las RIAs. JISBD 2009: 219-222

[7] Jim Conallen: Modeling Web Application Architectures with UML. Commun. ACM 42(10): 63-70 (1999)

[8] Request For Comments 2616 Hypertext Transfer Protocol 1.1

[9] Jesse James Garrett: Ajax: A New Approach to Web Applications http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

[10] Documentación de Oracle. http://www.oracle.com/technetwork/java/applets-137637.html

[11] Frames en W3C. http://www.w3.org/TR/html4/present/frames.html

[12] Adobe Flash Platform http://www.adobe.com/es/flashplatform/

[13] Adobe Flex. http://www.adobe.com/es/products/flex.html

[14] Request For Comments 46r The application/json Media Type for Javascript Object Notation.

[15] Página de JSON. http://www.json.org/

[16] Definición de DOM por W3C. http://www.w3.org/DOM/

[17] Web oficial de ExtJS4. http://www.sencha.com/products/extjs/

[18] Web oficial de Prototype. http://www.prototypejs.org/

[19] Web oficial de jQuery. http://jquery.com/
[20] Web oficial de YUI. http://developer.yahoo.com/yui/
[21] Web oficial de Sencha Labs. http://www.sencha.com/company/
[22] Web oficial de Dojo Toolkit. http://dojotoolkit.org/
[23] Web oficial de Script.aculo.es. http://script.aculo.us/
[24] Escritorios colaborativos online de Kohive. http://kohive.com/
[25] Ejemplo de código de ExtJs: http://dev.sencha.com/deploy/ext-4.0.0/examples/grid/binding-with-classes.html
[26] Documentación de MXML. http://learn.adobe.com/wiki/display/Flex/MXML
[27] ActionScript Technology Center. http://www.adobe.com/devnet/actionscript.html
[28] Creative Studio de HP. http://www.hp.com/hho/hp_create/
[29] Ejemplo de código de Flexhttp://www.adobe.com/devnet-apps/flex/tourdeflex/web/
[30] Web oficial de GWT. http://code.google.com/intl/es-ES/webtoolkit/
[31] Ofuscación de GWT. http://code.google.com/intl/es-ES/webtoolkit/doc/latest/FAQ_DebuggingAndCompiling.html
[32] Ejemplo de código de GWT: http://gwt.google.com/samples/Showcase/Showcase.html#!CwDataGrid
[33] Borrador de HTML 5 http://dev.w3.org/html5/spec/Overview.html
[34] Recursos HTML 5 (Google) http://www.html5rocks.com
[35] Recursos HTML 5 (Mozilla) https://developer.mozilla.org/es/HTML/HTML5
[36] Recursos HTML 5 (Microsoft) http://ie.microsoft.com/testdrive/Views/SiteMap/
[37] Recursos HTML 5 (Apple) http://www.apple.com/html5/
[38] Ejemplo de Canvas http://glge.org/demos/cardemo/
[39] CSS 3 en w3c http://www.w3.org/TR/CSS/#css3
[40] JavaServer Faces Technology. http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[41] JavaServer Pages Technology. http://www.oracle.com/technetwork/java/javaee/jsp/index.html
[42] JavaBeans Component API. http://docs.oracle.com/javase/7/docs/technotes/guides/beans/index.html
[43] The Vanguard Group, Inc. www.vanguard.com

[44] Ejemplo de código de JSF: http://facestutorials.icefaces.org/tutorial/dataTable-tutorial.html
[45] Gómez J., Cachero C., Pastor O. Conceptual Modeling of Device-Independent Web Applications. IEEE Multimedia, 8(2), 26–39, 2001.
[46] Eclipse Modelling Framework. http://www.eclipse.org/modeling/emf/
[47] Francisco Valverde, Oscar Pastor: Applying Interaction Patterns: Towards a Model-Driven Approach for RichInternet Applications Development, 2008
[48] Core J2EE Patterns http://www.corej2eepatterns.com/
[49] Antonio Navarro, Alfredo Fernández-Valmayor, Baltasar Fernández-Manjón, José Luis Sierra: Characterizing navigation maps for web applications with the NMM approach. Sci. Comput. Program. 71(1): 1-16 (2008)